UNIVERSITÉ DE MONTRÉAL

ACCÉLÉRATION D'UNE PLATEFORME D'ENCODAGE MPEG-4 À L'AIDE DE PROCESSEURS CONFIGURABLES

SIMON PROVOST
DÉPARTEMENT DE GÉNIE INFORMATIQUE
ÉCOLE POLYTECHNIQUE DE MONTRÉAL

MÉMOIRE PRÉSENTÉ EN VUE DE L'OBTENTION
DU DIPLÔME DE MAÎTRISE ÈS SCIENCES APPLIQUÉES
(GÉNIE INFORMATIQUE)
DÉCEMBRE 2006

Your file Votre référence
ISBN: 978-0-494-25567-4
Our file Notre référence
ISBN: 978-0-494-25567-4

UNIVERSITÉ DE MONTRÉAL

ÉCOLE POLYTECHNIQUE DE MONTRÉAL

Ce mémoire intitulé:

ACCÉLÉRATION D'UNE PLATEFORME D'ENCODAGE MPEG-4 À L'AIDE DE PROCESSEURS CONFIGURABLES

présenté par: PROVOST Simon

en vue de l'obtention du diplôme de: Maîtrise ès sciences appliquées

a été dûment accepté par le jury d'examen constitué de:

M. LANGLOIS Pierre, Ph.D., président

M. BOIS Guy, Ph.D., membre et directeur de recherche

Mme NICOLESCU Gabriela, Doct., membre et codirectrice de recherche

M. ABOULHAMID Mostapha, Ph.D., membre

REMERCIEMENTS

Je tiens tout d'abord à remercier mon directeur de recherche, M. Guy Bois, et ma codirectrice, Mme Gabriela Nicolescu pour leur support tout au long de cette recherche. Je désire également remercier Guy, et lui offrir mes sympathies, pour avoir dû utiliser une bonne partie de ses soirées et fins de semaine pour lire ce présent document.

Je me dois de dire un gros merci à Bruno Lavigueur pour l'immense aide qu'il a apporté tout au long de cette recherche. Ses innombrables questions, commentaires et suggestions ont grandement facilité l'avancement des travaux réalisés. Les contributions de Bruno à cette recherche ont largement dépassé toutes les attentes. Encore une fois merci.

Mes collègues de travail ont également tenu un rôle important lors de ces travaux de recherche. À cet égard, j'aimerais remercier (dans aucun ordre précis): J-F, Frank, Frank (vous déciderez entre vous lequel est lequel), Pat, Max, Cédric, Luc, Morti, David, Fouad, Nick et tous les autres membres du Circus. Ils ont su contribuer par des discussions intéressantes sur les projets de recherche, mais surtout par des activités moins constructives et hautement divertissantes. Grâce à eux, je sors de l'université riche de plusieurs amitiés supplémentaires.

Bien qu'ils n'aient pas contribué directement aux travaux de recherche, j'aimerais remercier plusieurs personnes pour leur aide dans des activités connexes. Tout d'abord, Alex et Réjean, nos deux techniciens du GRM pour leur excellent support et efficacité. Par la suite, Madeleine, Jeanne, Chantal, Marie-Yannick et Ghyslaine pour avoir été toujours si serviable.

Je tiens à remercier mes parents pour m'avoir toujours appuyés dans mes choix de carrière et m'avoir enduré chez eux toutes ces années. Finalement, j'aimerais remercier Chantal Perreault pour son support inconditionnel et pour avoir apporté une aide bien plus grande qu'elle ne saurait le croire.

RÉSUMÉ

Le marché des systèmes embarqués et systèmes sur puce modernes connaît une croissance impressionnante depuis quelques années. Les outils et techniques de conception classiques ne permettent plus de rencontrer les contraintes du marché en matière de coûts, de puissance de calcul et de délai de conception. Pour répondre à ces nouveaux défis, le domaine des systèmes sur puce a vu apparaître plusieurs techniques ou outils visant à accélérer le développement et augmenter la puissance de calcul tout en gardant les coûts faibles et un bon niveau de flexibilité.

Les processeurs configurables sont un de ces outils ayant vu récemment le jour. Cette nouvelle classe de processeurs, grâce à l'ajout d'instructions spécialisées, vise à offrir un niveau de performance rivalisant avec celui d'un couple processeur général et coprocesseur. Par contre, ces processeurs possèdent l'avantage d'être plus flexibles étant donné qu'ils permettent de maintenir une solution entièrement logicielle. De plus, les outils fournis avec ces processeurs offrent un soutien à la conception des instructions spécialisées et dans certains cas offrent même de les déterminer automatiquement, ce qui permet d'accélérer le délai de conception et réduit le risque d'erreurs. Une des principales contributions de cette recherche est de montrer comment les instructions spécialisées peuvent être utilisées pour réaliser des gains importants et de présenter les résultats obtenus pour une application d'encodage MPEG-4.

Un autre aspect vers lequel se sont penchés les concepteurs de systèmes sur puce est le développement de systèmes sur puce multiprocesseurs. Ces systèmes visent à atteindre la performance voulue en distribuant une application sur plusieurs processeurs. Malheureusement, l'ajout de plusieurs processeurs complexifie considérablement le développement du système à cause de tous les problèmes de synchronisation, de cohérence de données, de communication entre différents éléments hétérogènes et de

congestion. Dans le but de favoriser la conception de tels systèmes, des outils ont été développés afin d'élever le niveau d'abstraction lors de la conception de plateformes, masquant ainsi plusieurs des problèmes reliés aux plateformes multiprocesseurs. Par exemple, la bibliothèque SystemC permet de modéliser un système complet, comme le fait VHDL, mais en utilisant le langage C++. Il existe également des plateformes de développement à haut niveau d'abstraction permettant de facilement brancher, simuler et faire la synthèse de plusieurs composants d'une plateforme et qui automatisent une bonne partie du processus de conception d'un système sur puce multiprocesseurs.

La deuxième contribution principale de cette recherche est d'utiliser une plateforme de développement à haut niveau d'abstraction, nommée StepNP, pour simuler un système sur puce multiprocesseurs utilisant des processeurs configurables. Nous montrerons qu'un tel système permet d'obtenir des gains en performance très intéressants et que la puissance de calcul croît presque linéairement avec le nombre de processeurs en l'absence de contention.

ABSTRACT

Modern embedded systems and systems on chip have known an impressive growth in the last few years. Traditional tools and design techniques are becoming insufficient to meet the constraints of the market in regard to costs, processing power and design time. To answer these new challenges, the field of systems on chip saw the appearance of several techniques or tools aimed at accelerating the development cycle and increasing the processing power while keeping the costs low and a good level of flexibility.

Configurable processors are one of these tools which appeared recently. This new class of processors, thanks to the addition of specialized instructions, aims at offering a level of performance competing with that of a general processor coupled with a coprocessor. However, these processors also have the advantage of being more flexible since they make it possible to maintain an entirely software solution. Moreover, the tools provided with these processors offer a support to the design of the specialized instructions and in certain cases even offer to determine them automatically, which makes it possible to accelerate the design time and reduces the risk of errors. One of the main contributions of this research is to show how the specialized instructions can be used to obtain important speedup and to present results obtained for an MPEG-4 encoding application.

Another aspect towards which systems on chip designers are leaning is the development of multiprocessors systems on chip. These systems aim at reaching the performance needed by distributing an application over several processors. Unfortunately, the addition of several processors complexes considerably the development of the system because of all the problems related to synchronization, data coherency, communication between various heterogeneous elements and congestion. In the hope of supporting the design of such systems, tools were developed to raise the level of

abstraction when designing platforms, thus masking several of the problems related to multiprocessors platforms. For example, the SystemC library makes it possible to model a complete system, as with VHDL, but by using the C++ language. There are also high abstraction level development platforms which allow to easily connect, simulate and make the synthesis of several components of a platform and which automate a good part of the development process of a multiprocessors system on chip.

The second main contribution of this research is to use a high level development platform, named StepNP, to simulate a multiprocessors system on chip using configurable processors. We will show that such a system makes it possible to obtain very interesting performance speedups and that the computing power scales very well with the number of processors.

TABLE DES MATIÈRES

LISTE DES FIGURES

LISTE DES ACRONYMES

AMBA	Advanced Microcontroller Bus Architecture
ANSI	American National Standards Institute
API	Application Programming Interface
APU	Auxiliary Processing Unit
ArchiFlex	Architecture Flexible
ASIC	Application Specific Integrated Circuit
ASIP	Application Specific Instructions Set Processor
BCA	Bus Cycle Accurate
CE	Concurrency Engine
CAO	Conception Assistée par Ordinateur
CIL	Common Intermediate Language
CORBA	Common Object Request Broker Architecture
DCOM	Distributed Component Object Model
DCT	Discrete Cosine Transform
DSP	Digital Signal Processing
DFG	Dataflow Graph
DSOC	Distributed System Object Component
DSP	Digital Signal Processing
ESL	Electronic System Level
FLIX	Flexible-Length Instructions Extensions
FPGA	Field Programmable Gate Array
GIPS	Giga-Instructions Par Secondes
GPP	General Purpose Processor
HDL	Hardware Description Language
HW	Hardware
IC	Integrated Circuit

IMC	Interface Method Call
ISA	Instruction Set Architecture
ISEF	Instruction Set Extension Fabric
ISS	Instruction Set Simulator
LISA	Language for Instruction Set Architecture
MAC	Multiply And Accumulate
MESCAL	Modern Embedded Systems, Compilers, Architectures and Languages
MPE	Message Passing Engine
MPEG-4	Motion Picture Expert Group 4
MPSoC	Multiprocessor System on Chip
NoC	Network on Chip
NOP	No Operation
OCP	Open Core Protocol
ORB	Object Request Broker
OSCI	Open SystemC Initiative
PE	Processing Element
RISC	Reduced Instruction Set Computing
RPC	Remote Procedure Call
RLE	Run Lenght Encoding
RTL	Register Transfer Level
RTOS	Real Time Operating System
SAD	Sum of Absolute Differences
SCIDE	SystemC Integrated Development and Debug Environment
SIMD	Single Instruction Multiple Data

SISSMA	SPACE Engine for Synchronization of SystemC Multiprocessor / Multithreaded Applications
SoC	System on Chip
SocGen	SoC Generation
SocMon	SoC Monitor
SOCP	SystemC Open Core Protocol
SMP	Symmetric Multiprocessing
SPACE	SystemC Partitioning of Architectures for Codesign of Embedded Systems
StepNP	System-Level Exploration Platform for Network Processors
SW	Software
TF	Timed functionnal
TIE	Tensilica Instruction Extension
TLM	Transaction Level Modeling
TTL	Task Transaction Level
UAL	Unité Arithmétique et Logique
UTF	Untimed Functionnal
VCC	Virtual Component Co-Design
VHDL	VHSIC Hardware Description Language
VHSIC	Very High Speed Integrated Circuit
VLIW	Very Long Instruction Word
XLMI	Xtensa Local Memory Interface
XPRES	Xtensa processor extension synthesis
XTMP	Xtensa Modeling Protocol

LISTE DES TABLEAUX

LISTE DES ANNEXES

INTRODUCTION

Le marché des systèmes embarqués, depuis quelques années, ne cesse de croître. On doit cette avancée en grande partie à la loi de Moore, établie en 1965 par Gordon Moore co-fondateur de IBM, qui prédisait à l'époque que le nombre de transistors sur une même puce augmenterait de 50% par année. Quarante ans plus tard, cette loi tient toujours la route et les développeurs de circuits intégrés (IC, *Integrated Circuit*) travaillent d'arrache pied pour maintenir cette croissance. L'intégration d'un nombre toujours grandissant de composants électroniques sur une simple puce, qu'on appel système sur puce (SoC, *System On Chip*), permet aux compagnies de systèmes embarqués de séduire leurs clients avec de plus en plus de fonctionnalités. On n'a qu'à regarder le marché des téléphones cellulaires pour voir l'effet de cette croissance. Il n'y a pas si longtemps, un téléphone ne servait, sans aucune surprise, qu'à téléphoner. Aujourd'hui, ils permettent d'observer son interlocuteur, de prendre des photos et films, d'envoyer et recevoir des courriels et une panoplie d'autres caractéristiques intéressantes. Avec la barrière de un milliard de transistors à l'horizon [13], il y a fort à parier que cette croissance n'est pas finie.

Une technique utilisée récemment, et rendue possible grâce à l'augmentation de la densité de transistors sur une puce, pour combler les demandes toujours plus grandes en puissance de calcul est la combinaison de plusieurs processeurs sur une même puce. Cette technique à fait naître le concept de systèmes sur puce multiprocesseurs (MPSoC,*Multiprocessor System on Chip*). Les MPSoC, par contre, sont plus qu'un simple système visant à augmenter la puissance de calcul par l'ajout de processeurs [36]. Ceux-ci sont des architectures complexes réunissant ensemble un grand nombre de composants dédiés et hétérogènes, comme différents types de processeurs, de mémoires et de canaux de communication.

Un autre élément important ayant récemment fait son entrée dans le domaine des systèmes embarqués est l'utilisation d'unités de traitement configurables. Ce nouveau type d'unité de calcul présente principalement les mêmes avantages qu'un processeur d'usage général (GPP, *General Purpose Processor*) mais possède en plus la possibilité d'étendre le jeu d'instruction avec des instructions spécialisées et de modifier certains paramètres architecturaux. Avant cette venue, le traitement d'une application était séparé entre des partitions logicielles, lentes mais plus facilement programmables et plus flexibles, et des partitions matérielles, rapides mais complexes à développer. Maintenant, grâce aux unités de traitement configurables, les concepteurs de SoC disposent d'une alternative à mi chemin entre les deux.

Problématique

Bien que la loi de Moore suit son cours, les développeurs de systèmes embarqués sont confrontés à bien d'autres problèmes que l'intégration de plusieurs transistors sur une même puce. Comme le montre la figure 1, la productivité des développeurs, liée en grande partie aux outils de programmation, ne croît pas au rythme de la loi de Moore, ce qui crée un écart de productivité important qui ne cesse de grandir avec les années. Le besoin d'avoir des outils de développement adéquats est illustré par le fait que ceux-ci soient le facteur numéro un influençant le choix d'un produit plutôt que celui d'un compétiteur [60].

Par le passé, pour atteindre les nouveaux standards de performance et rencontrer les délais de mise en marché, les concepteurs de systèmes ont toujours dû élever le niveau d'abstraction de leurs designs. Ainsi, nous somme passé de programmes écrits en assembleur à des langages haut niveau (e.g. C, Java), des dessins en portes logiques aux RTL (*Register Transfer Level*) (e.g. VHDL, Verilog). Nous en sommes arrivé à un point où le marché devra à nouveau élever ce niveau d'abstraction pour tenir en

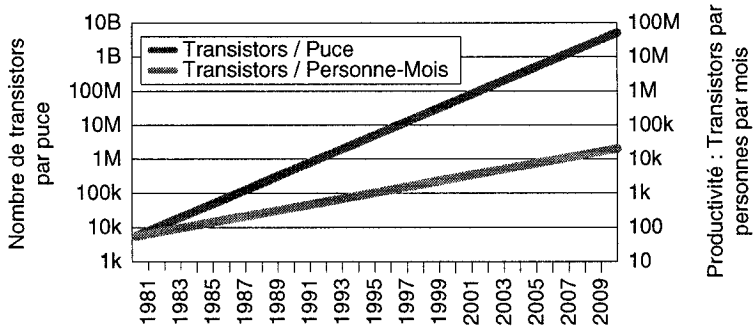

Figure 1 Écart de productivité

compte les nouveaux défis posés par les MPSoC.

Un des problèmes auxquels sont confrontés les développeurs de systèmes embarqués est au niveau de l'intégration des composants. L'approche classique pour la conception de SoC consiste à réaliser la partie matérielle du système en premier et la partie logicielle ensuite. Cette approche comporte son lot d'inconvénients. Premièrement, la difficulté, et par conséquent le délai, liée à la conception du logiciel croit exponentiellement avec les années. Cette situation vient en grande partie de la difficulté à distribuer et synchroniser adéquatement l'application sur l'ensemble des processeurs du système. La conception du logiel a d'ailleurs remplacé la consommation de puissance au premier rang des facteurs nuisant à la continuation de la loi de Moore [31]. Deuxièmement, il est fort probable que la conception et la réalisation de la partie logicielle révèle des failles non anticipées dans la partie matérielle, ce qui peut nécessiter un retour en arrière et des retards importants [33]. Troisièmement, cette approche ne permet pas de tester plusieurs options de partitionnement puisque la partie matérielle précède la partie logicielle. Un autre problème relié à l'approche classique de conception est que les spécifications d'un système à concevoir sont décrites dans une langue native (e.g. anglais, français) et sont généralement très lourdes. Malgré tous les efforts pour les rendre les plus claires possible, ces spécifications restent généralement floues et sont la plus grande source

d'erreurs dans le design [44]. Finalement, pour tenir compte de l'arrivé des unités de traitement configurables dans le choix des composants, il est nécessaire de se doter d'outils permettant d'exploiter la conception d'instructions spécialisées.

Pour contrer les problèmes mentionnés, l'industrie a récemment vu l'apparition du concept d'ESL (*Electronic System Level*). La notion exacte d'ESL n'est pas encore bien définie, mais elle implique entre autre la modélisation des communications logicielles / matérielles, la conception au niveau système et l'élévation du niveau d'abstraction [40]. Cette nouvelle approche de conception possède plusieurs avantages considérables :

- Le modèle haut niveau constitue la spécification du système. Puisque celle-ci est écrite dans un langage exécutable / simulable, ceci permet d'éliminer toute ambiguïté et permet de vérifier plus aisément que l'implémentation correspond aux spécifications [12].

- Le développement de la partie logicielle peut débuter plus rapidement étant donné qu'un modèle de la partie matérielle est disponible plus tôt dans le processus de développement.

- Cette approche favorise une meilleure exploration architecturale. Un modèle décrit à un haut niveau d'abstraction est plus rapide à simuler qu'un modèle à bas niveau. De plus, comme le système entier est initialement décrit dans un seul langage, cela implique que le partitionnement logiciel / matériel n'est pas fixe. Ceci permet donc aux concepteurs de rapidement évaluer plusieurs possibilités d'architecture ou de partitionnement et de raffiner la meilleure solution pour le système.

- Le niveau système se prête bien à l'exploration des diverses possibilité offertes par les processeurs configurables puisque l'architecture n'est pas encore fixée [41].

Objectifs

L'objectif principal de ce travail consiste à évaluer l'avantage offert par les processeurs configurables dans un système embarqué multiprocesseurs requérant une grande puissance de calcul. L'approche classique pour atteindre le niveau de performance requis consiste à utiliser plusieurs processeurs couplés de coprocesseurs dédiés. Nous visons à mesurer les gains qu'il est possible d'obtenir en utilisant des instructions spécialisées plutôt que des accélérateurs matériels conventionnels. De plus, la conception se fera à un haut niveau d'abstraction, permettant d'évaluer la facilité avec laquelle il est possible de modéliser un MPSoC complet. Bien que l'implémentation physique du circuit modélisé ne fasse pas partie des objectifs, le modèle haut niveau permettra d'obtenir des conclusions intéressantes sur l'utilisation des processeurs configurables dans un MPSoC.

Méthodologie

Pour montrer l'atteinte des objectifs, nous utiliserons une application d'encodage MPEG-4 (*Motion Picture Expert Group 4*) comme cas de test. Cette application fut choisie puisqu'elle est très exigeante en puissance de calcul.

La méthodologie utilisée consistera à accélérer l'application en question à l'aide de deux grandes étapes. Premièrement, nous étendrons le jeu d'instruction standard d'un processeur configurable avec des instructions spécialisées, procurant un premier niveau d'accélération. Dans un deuxième temps, nous distribuerons l'application sur plusieurs processeurs configurables pour obtenir un gain supplémentaire. Par la suite, nous comparerons la vitesse d'encodage, en images par secondes, avec celle obtenue à l'aide d'une plateforme utilisant des processeurs d'usage général et des coprocesseurs dédiés.

Contributions

La réalisation de ce travail entraîne deux principales contributions. En premier lieu, nous montrons comment il est possible d'obtenir des gains en performance très intéressants, pour une application d'encodage vidéo, en utilisant les processeurs configurables et particulièrement les instructions spécialisées. La deuxième contribution est l'intégration d'un modèle simulable du processeur Xtensa à l'intérieur de la plateforme de simulation haut niveau StepNP. La version de StepNP utilisée lors de cette recherche est plus récente que celle utilisée par Bruno Lavigueur [38]. On y inclu entre autre plusieurs nouvelles fonctionnalités visant principalement les MPSoC.

Organisation du mémoire

Le reste du mémoire est divisé comme suit. Le chapitre 1 passe en revue un éventail des langages de conception au niveau système, des plateformes de développement à haut niveau d'abstraction et des processeurs configurables. Le chapitre 2 explique les techniques utilisées pour étendre le jeu d'instruction du processeur configurable et présente les résultats obtenus pour l'accélération d'une application d'encodage MPEG-4. Le chapitre 3 présente en détail la plateforme StepNP et les opérations nécessaires pour y intégrer un modèle du processeur Xtensa. Le chapitre 4 montre la plateforme multiprocesseurs d'encodage MPEG-4 réalisée à l'aide de StepNP et donne les résultats obtenus suite à l'ajout de processeurs configurables.

CHAPITRE 1

REVUE DE LA CONCEPTION AU NIVEAU SYSTÈME ET DES PROCESSEURS CONFIGURABLES

Comme il est expliqué dans l'introduction, ce chapitre présente un aperçu de diverses techniques récentes utilisées pour la conception de MPSoC. On y décrit dans un premier temps la conception au niveau système, soit plus précisément les langages qui l'accompagnent et les plateformes de développement disponibles. Dans un deuxième temps, on présente un éventail de processeurs configurables.

1.1 Conception au niveau système

Pour résumer, la conception au niveau système est une élévation du niveau d'abstraction pour le design de systèmes embarqués. Comme il est expliqué dans l'introduction, ceci comporte plusieurs avantages notables et est nécessaire pour contrer les nouveaux problèmes apportés par les systèmes multiprocesseurs.

1.1.1 Langages de design au niveau système

Les langages utilisés lors du développement classique d'un système embarqué sont mal adaptés pour le développement au niveau système. Les langages de conception matérielle comme VHDL ou Verilog sont évidemment adéquat pour le raffinement de la partie matérielle. Par contre, ces langages ne permettent pas de décrire le fonctionnement du système à un haut niveau d'abstraction. D'un autre côté, les langages de programmation comme le C/C++ permettent de décrire toute la

fonctionnalité d'un système à un haut niveau d'abstraction, mais ils ne sont pas bien adaptés pour modéliser les communications logicielles/matérielles et les composantes matérielles. De plus, ils ne permettent pas le raffinement vers un niveau d'abstraction plus bas. Pour combler ce problème, plusieurs langages ont étés proposés pour permettre la conception au niveau système. Ces langages possèdent une combinaison de constructions propres aux langages de programmation haut niveau et aux langages de conception matérielle.

Le principal intérêt de ces langages est qu'ils permettent de conserver le même langage tout au long des étapes de raffinement. Comme mentionné précédemment, ceci permet d'évaluer plus aisément différents types de partitionnement. Autre avantage important, l'utilisation d'un langage unique permet de conserver les mêmes tests à chacune des différentes étapes et limite le risque d'erreurs liées à porter le système d'un langage à un autre.

1.1.1.1 SystemC

SystemC fut développé et est maintenu par le consortium OSCI (*Open SystemC Initiative*), regroupant plusieurs entreprises telles Mentor, Coware, Synopsys et autres. SystemC n'est pas un nouveau langage à proprement parler, mais plutôt une bibliothèque de fonctionnalités ajoutées au populaire langage C++, et c'est d'ailleurs là une de ses principales forces. La bibliothèque offre deux principaux intérêts pour la conception au niveau système. Premièrement, on y retrouve plusieurs types d'éléments propres à la conception matérielle et deuxièmement, elle offre un noyau de simulation similaire à celui qu'on retrouve dans d'autres langages de conception matérielle tels VHDL [47].

Plusieurs éléments de modélisation en SystemC ont étés empruntés d'autres HDL (*Hardware Description Language*). On retrouve, premièrement, le concept de

modules, comparable à une entité en VHDL, qui se traduit sous la forme d'une classe en C++ de laquelle hérite tous les modules du système. Ces modules peuvent également être créés de manière hiérarchique, permettant l'assemblage de systèmes complexes à partir de composants simples. La bibliothèque ajoute également les processus et la liste de sensibilité qui leur est associée, émulant ainsi le comportement concurrent inhérent aux systèmes matériels. Les processus peuvent être également synchronisés sur des événements ou par des délais temporels. Un ajout intéressant apporté par la version 2.1 de SystemC est que ces processus peuvent être instanciés de manière statique, comme en VHDL, ou de manière dynamique en cours d'exécution, procurant ainsi plus de flexibilité au concepteur.

Un élément important apporté pas SystemC, et autres langages de conception au niveau système, est l'abstraction des communications entre les modules. On y introduit la notion de ports représentant les connections à un bas niveau d'abstraction. On y introduit également les canaux de communication et les interfaces, permettant de modéliser à un haut niveau d'abstraction les communications entre modules. Cette fonctionnalité permet de modéliser des protocoles de communication plus complexe comme ceux offerts par les réseaux sur puce (NoC, *Network on Chip*), qui sont devenu très populaires ces dernières années [9]. Les interfaces sont représentées par une fonction C qui peut être appelée par un port qui lui est connecté et qui peut passer n'importe quel type de donnée comme argument, comme par exemple une structure contenant les divers paramètres nécessaires, élevant ainsi le niveau d'abstraction. Ce type de communication est aussi appelé niveau transactionnel (TLM, *Transaction Level Modeling*) puisqu'on se concentre sur le passage de paquets en faisant abstraction des détails du protocole [14].

L'opération de test en SystemC constitue également un avantage d'utiliser ce langage. SystemC possède toute les fonctionnalités du langage C/C++, permettant ainsi la réalisation de bancs d'essais plus complexes que ce qu'il est possible en VHDL ou

en Verilog [24]. Ce banc d'essai prend la forme d'un programme C/C++ exécutable dans lequel on instancie et connecte nos différents modules.

Le noyau de simulation utilisé dans SystemC, basé sur un mécanisme d'évaluation et de mise à jour, est très similaire à celui qu'on retrouve dans VHDL et autres HDL. Le concept de delta cycles, fréquemment utilisé dans les langages de description matérielle, y est également présent. Le processus de simulation possède cinq grandes étapes qui sont brièvement décrites ci-dessous [46].

1 - Initialisation : Au début de la simulation, le système est initialisé et tous les processus sont exécutés, à l'exception de ceux sensibles à l'horloge ou que le programmeur a spécifié explicitement comme non initialisés.

2 - Évaluation : Le système exécute successivement tous les processus présents dans la liste des processus prêts. Le noyau de SystemC étant non préemptif, lorsqu'un processus est exécuté, il l'est jusqu'à ce qu'il se suspende lui-même suite à un appel à l'instruction `wait`, qui peut être explicite ou implicite du point de vue du programmeur. Encore une fois pour imiter le langage VHDL, les signaux ne sont pas mis à jours instantanément mais plutôt cédulés pour l'être à la phase 3.

3 - Mise à jour : Les signaux sont mis à jour suite aux changements ayant étés apportés à l'étape 2.

4 - Notification d'un delta cycle : Dans la cas où un processus aurait demandé une attente ou une notification avec un délai d'un delta cycle (e.g. en utilisant `wait(0)`), on détermine la liste des processus sensibles à cet événement qui formeront la liste de processus prêts, et on retourne à l'étape 2.

5 - Notification d'un délai : Le temps de simulation est avancé jusqu'au prochain événement, la liste des processus prêts est déterminée et on passe à l'étape 2. Lorsqu'il n'existe aucun événement dans le futur, la simulation est terminée.

1.1.1.2 SystemVerilog

Comme son nom l'indique, ce langage est une extension du très populaire langage de conception matérielle Verilog. À l'origine, les développeurs de ce langage avaient deux objectifs principaux lors de sa conception [28]. Le premier était d'étendre les fonctionnalités de conception de Verilog, en partie en empruntant certaines des fonctionnalités présentes en VHDL. Le deuxième était d'augmenter le support offert par Verilog au niveau de la vérification.

Parmi les ajouts présents dans SystemVerilog, on retrouve plusieurs types de donnés pour mémoriser les entiers, points flottants ou signaux logiques. On y retrouve également le type Chandle (*C handle*) qui s'apparente à un pointeur sécuritaire tel qu'on les retrouve dans certains langages de programmation haut niveau. SystemVerilog possède également des types de données utilisées lors de la synchronisation interprocessus. Premièrement, on retrouve les événements, qui contrairement à SystemC, sont actifs pendant tout le cycle de simulation. Ainsi, lorsqu'un processus déclenche un événement, si un autre processus se met en attente de celui-ci par la suite mais au même cycle, le processus reprendra son exécution immédiatement. Outre les événements, SystemVerilog possède deux autres types de données de synchronisation soit les boîte de messages et les sémaphores. Pour favoriser la modélisation à un haut niveau d'abstraction et la réalisation de bancs d'essais, on a également ajouté la programmation orientée objet. SystemVerilog supporte la majorité des concepts reliés à ce type de programmation comme illustré ci-dessous :

- Encapsulation - Héritage simple - Classes abstraites - Classes paramétrées
- Pointeur *this* - Surcharge - Polymorphisme - Méthodes virtuelles
- Variables et méthodes statiques

On retrouve également plusieurs concepts dans SystemVerilog analogues à ceux de SystemC. Notamment, SystemVerilog possède les interfaces, permettant de séparer

les communications des modules, et permet de créer dynamiquement des processus.

La principale force de SystemVerilog vient de son support à la vérification et particulièrement des assertions. Premièrement, la syntaxe utilisée pour les assertions est la même que pour le design, permettant ainsi de les inclure directement dans le code aux endroits appropriés. Ensuite, on nous garantit que les assertions sont construites de manière à produire le même résultat lors de la simulation, qui est basée sur les événements, que lors d'une vérification formelle par des outils externes, qui est basée sur les cycles d'horloge [27]. Ceci est réalisé en échantillonnant les signaux impliqués dans les assertions à chaque cycle d'horloge lors de la simulation.

La nouveauté introduite par SystemVerilog est l'assertion concurrente qui permettent de décrire le comportement du système dans le temps. Pour définir ce comportement, on utilise les expressions séquentielles. Ceux-ci permettent de spécifier des expressions booléennes accompagnées de leur relation temporelle. Par exemple, si A et B sont des expressions booléennes, il serait possible de spécifier que A doit se produire un cycle après B. On peut également combiner les expression séquentielles ensembles pour spécifier qu'une séquence doit être inclue dans une autre, que deux séquences doivent se terminer en même temps ou encore que la fin d'une séquence doit entraîner le départ d'une autre. Un autre attrait très intéressant de SystemVerilog pour la vérification est la génération aléatoire de contraintes, qui est inclus à la programmation orientée objet. Ainsi, les contraintes sur un attribut d'une classe sont spécifiées à l'intérieur de la classe même. Au moment d'instancier la classe en question, l'utilisateur n'a qu'à indiquer qu'il veut générer aléatoirement les valeurs des paramètres et l'outil va déterminer une solution satisfaisant les contraintes données. Le solutionneur va toujours trouver une valeur respectant les contraintes s'il en existe une [27]. Une classe qui hérite d'une autre hérite également de toutes les contraintes qui lui sont associées, favorisant la réutilisation et le raffinement des contraintes pour des cas plus spécifiques.

1.1.1.3 Autres langages

Il existe bien d'autres langages permettant de faire de la conception de SoC au niveau système dont quelques uns sont décrits ci-dessous.

SpecC [29]: Originalement développé à l'université de Californie, SpecC est l'un des premiers langages de conception au niveau système à avoir vu le jour. Bien que la syntaxe soit basée sur celle du ANSI-C, il s'agit bien d'un nouveau langage et non d'une simple bibliothèque comme dans le cas de SystemC. L'objectif des concepteurs était de définir un langage permettant d'intégrer les spécifications et les premières phases de design lors de la conception d'un SoC. Comme pour la plupart des langages de conception au niveau système, un des principaux avantages de SpecC est de séparer les communications entre modules des modules eux-mêmes. SpecC permet de définir le comportement des différents modules en utilisant une classe nommée behavior et ces modules peuvent être connectés ensemble via des interfaces ou encore regroupés ensemble pour former des modules hiérarchiques. SpecC supporte quatre types d'exécution des modules hiérarchiques :

Séquentielle : Dans ce type d'exécution, les modules sont tous exécutés un à la suite de l'autre. Ce mode permet de simuler un comportement logiciel.

Concurrent : Tous les modules sont exécutés de manière concurrente, permettant de modéliser un comportement matériel.

Pipeline : Le mode pipeline permet d'exécuter plusieurs modules de manière concurrente, mais comme son nom l'indique, de manière pipelinée. Plus concrètement, le mode pipeline prend la forme d'une boucle *for* où chaque module représente un étage du pipeline. À la première itération, seulement le premier étage du pipeline est exécuté. À la seconde, les deux premiers étages sont exécutés en parallèle, et ainsi de suite jusqu'à ce que le pipeline soit rempli.

Lorsque la condition de la boucle *for* est fausse, n-1 (n étant le nombre d'étages) itérations supplémentaires sont exécutées pour vider les étages du pipeline. Des variables de type *file* permettent également le passage de valeurs entre les étages du pipeline.

Machine à états : SpecC permet également de modéliser le comportement d'une machine à états. Dans ce mode, chaque module représente un état de la machine et le test de certaines conditions de sortie des modules permet de spécifier quel sera le module suivant à être exécuté.

ESys.Net [37]: ESys fut développé à l'université de Montréal par le groupe de recherche de M. Mostapha Aboulhamid. Le but premier de ce langage était de couvrir les lacunes présentes dans SystemC, comme le manque de support au déverminage et la difficulté de modéliser la partie logicielle d'un SoC. Alors que SystemC est une bibliothèque C++, c'est à l'aide de C# que le développement de ESys.Net fut réalisé. L'utilisation de .Net pour le développement d'un langage de conception à haut niveau offre plusieurs avantages soient :

- La gestion des processus, incluant le contrôle de ceux-ci et la préemption, permet de mieux modéliser le comportement de la partie logicielle.
- La gestion automatique de la mémoire allège la tâche des programmeurs et réduit le risque d'erreur.
- La présence d'un format intermédiaire commun (CIL, *Common Intermediate Language*) permet de faire du développement indépendant de la plateforme et du langage. De plus, ce format intermédiaire peut être utilisé comme entrée par les outils de conception assistés par ordinateur (CAO).
- Le mécanisme de réflexion présent dans .Net permet de construire plus aisément l'hiérarchie du système et favorise l'introspection de la plateforme sous test.

- Les interfaces font partie des fonctionnalités de base de .Net, simplifiant la conception des canaux de communication.
- Les services Web offerts par .Net permettent à plusieurs équipes de travailler conjointement au développement.
- L'ajout d'étiquettes donne la possibilité de spécifier plus aisément certaines directives d'exécution, comme par exemple les différents processus, listes de sensibilité, initialisation.

Rosetta [4]: Ce nouveau langage de modélisation tire son nom de la pierre de Rosette, ayant permis de déchiffrer les hiéroglyphes égyptiens. L'approche utilisée par Rosetta pour la modélisation diffère sensiblement des autres langages illustrés précédemment. Un problème rencontré lors de la modélisation est que les diverses parties d'un système hétérogène sont généralement mieux décrites en utilisant une certaine sémantique qui leur est propre. Par exemple, un système combinant des éléments mécaniques, optiques et électriques analogiques / numériques pourrait présenter un tel problème. Ou encore, la modélisation des contraintes d'un système qui peut combiner la puissance, la surface et le délai. Le langage Rosetta permet de définir un système par une combinaison de plusieurs facettes qui illustrent chacune une perspective du système. Chaque facette est associée à un domaine spécifique (e.g. mécanique, machine à états, temps discret) qui fournit une sémantique pour ce domaine. La force de Rosetta est qu'il permet de définir les interactions entre les domaines, ou encore comment traduire les facettes d'un domaine à un autre. Cette capacité permet donc de spécifier toutes les parties et contraintes d'un système en utilisant la sémantique la mieux adaptée pour chacune. Par la suite, ces différentes parties et contraintes sont combinées ensemble pour former un système complet. Ceci fait donc de Rosetta un langage très puissant lors de la conception de systèmes hétérogènes.

1.1.2 Plateformes de développement

1.1.2.1 SPACE

Développé à l'École Polytechnique de Montréal, SPACE (*SystemC Partitioning of Architectures for Codesign of Embedded Systems*) a pour but premier de combler les lacunes de SystemC en matière de simulation de la partie logicielle d'un système embarqué [10]. Étant donné que tous les processus créés en SystemC sont exécutés de manière concurrente, ou simulés comme tel, ceci représente mal le comportement de processus exécutés sur un processeur puisque ceux-ci devraient être exécutés séquentiellement. De plus, les processus de SystemC ne supportent pas la préemption ou l'ordonnancement selon la priorité, qui sont des éléments souvent requis par les systèmes embarqués. Pour résoudre ce problème, SPACE intègre un système d'exploitation temps réel (RTOS, *Real Time Operating System*) à une plateforme SystemC. Le second objectif de SPACE est de faciliter le partitionnement logiciel / matériel. Tous les modules suivent des règles de conception qui permettent de les déplacer d'une partition logicielle à une partition matérielle, ou vice-versa, sans aucune modification du module lui-même. Pour ce faire, SPACE fournit une interface générique aux modules qui redirige les appels de fonction vers ceux de SystemC ou du RTOS dépendamment si le module est sur une partition matérielle ou logicielle. Finalement, SPACE fournit une méthodologie de conception pour les systèmes embarqués basée sur le raffinement en trois niveaux successifs. Ces trois niveaux sont décrits ci-dessous et résumés à la figure 1.1 [19].

Le premier niveau, L1, permet de rapidement valider le fonctionnement de l'application. À cette étape, aucune distinction n'est faite entre les modules logiciels et matériels, ils sont tous représentés par des modules SystemC de base. De plus, tous ces modules sont connectés ensemble par un canal de communication au niveau transactionnel sans délai, ce qui permet de simuler très rapidement.

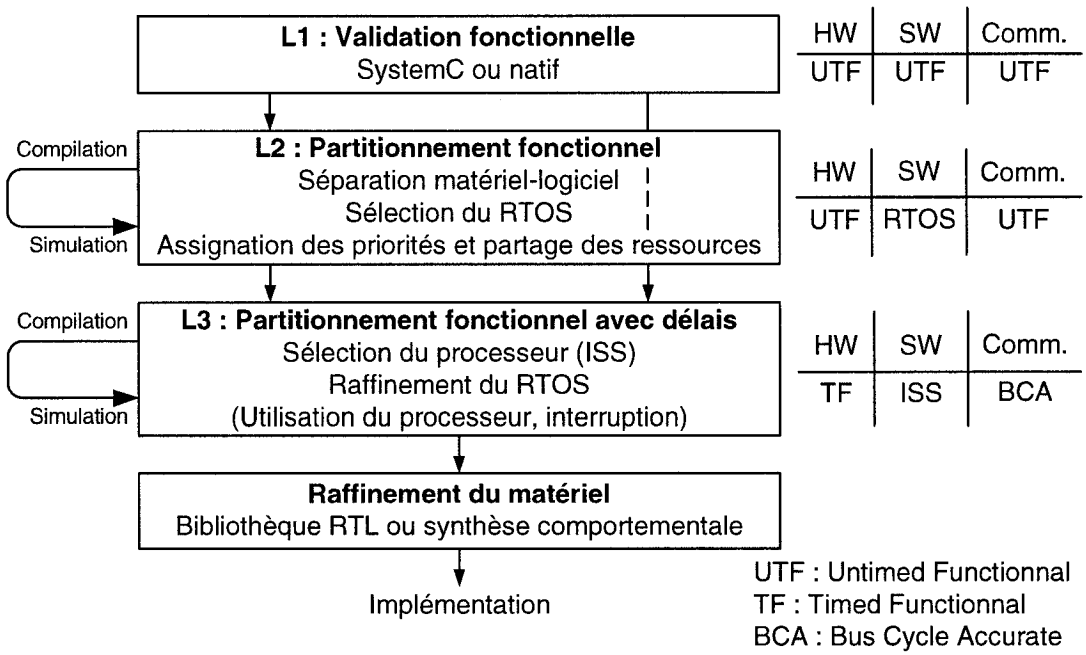

Figure 1.1 Niveaux de raffinement de SPACE

Le deuxième niveau, L2, donne une première approximation du partitionnement logiciel / matériel. À ce stade, les partitions logicielles sont prises en charge par un RTOS (*Real Time Operating System*) (i.e. MicroC/OS-II) qui exécute sur la machine hôte alors que le canal de communication et les partitions matérielles restent inchangés. Ce niveau permet de vérifier le réglage des priorités et de s'assurer qu'il n'y a pas d'interblocage. Tous les modules logiciels sont exécutés sur la machine hôte qui utilise l'interface générique pour rediriger les appels de fonctions vers le RTOS. Afin de communiquer avec la partie matérielle du système, incluant la mémoire principale, la partie logicielle est munie d'une couche qui communique par interface de connexion (*socket*) avec la partie matérielle du système. Puisque plusieurs détails architecturaux ne sont pas encore fixés (e.g. processeurs, communications), le niveau L2 reste beaucoup plus rapide à simuler que le niveau L3. Une amélioration en cours vise à éliminer la communication par interface de connexion et fournir une première approximation des délais de la plateforme dès le niveau L2. Cette simulation se fait sans simuler au niveau du cycle, ce qui est l'objectif du niveau L3 et nécessite plus

de temps de simulation. Les détails d'implémentation du nouveau L2, par contre, ne sont pas connus à ce moment.

Le troisième niveau, L3, implémente plusieurs détails architecturaux. Premièrement, on y introduit un simulateur de jeu d'instruction (ISS, *Instruction Set Simulator*) du processeur cible (e.g. ARM, Microblaze) sur lequel seront exécutés le RTOS et les modules logiciels. Cet ISS est encapsulé dans un module SystemC et relié au canal de communication de la plateforme. Le canal de communication est remplacé par un modèle précis au cycle près et des délais son ajoutés aux modules matériels. La plateforme obtenue au niveau L3 permet d'obtenir des délais relativement précis tout en étant plus rapide à simuler qu'un model RTL, ceci procure donc la possibilité de tester plusieurs choix architecturaux et de raffiner la meilleure solution trouvée.

SPACE fournit également un outil de gestion de la concurrence, nommé SISSMA (*SPACE Engine for Synchronization of SystemC Multiprocessor/Multithreaded Applications*), permettant de contrôler l'accès aux zones partagées [55]. L'outil SISSMA fournit un module matériel, connecté sur le canal de communication de la plateforme, qui centralise la gestion des sémaphores partagés par plusieurs éléments de calcul. Ainsi, la prise d'un sémaphore se fait par un simple accès mémoire au module SISSMA, ce qui assure l'atomicité des opérations. Il est également possible d'utiliser des sémaphores locaux dans le cas où ceux-ci ne sont utilisés que par un seul processeur. Pour ce deuxième type de sémaphore, SISSMA réutilise ceux incluent à MicroC/OS-II.

1.1.2.2 Virtual Platform Designer

Cette suite d'outils de la compagnie CoWare fournie un éventail complet de fonctionnalités permettant de créer des plateformes virtuelles basées sur la bibliothèque SystemC. Virtual Platform Designer est une combinaison de plusieurs outils inter

reliés dont Model Libraries, Processor Designer et Platform Architect[1], qui sont décrits ci-dessous [1].

1) **Model Libraries.** Cet outil de développement donne accès à une bibliothèque de composants, décrits en SystemC, parmi lesquels on retrouve des processeurs, mémoires, bus de communication et périphériques. CoWare a travaillé en collaboration avec ARM, MIPS et LSI affin d'inclure à la bibliothèque un ISS pour leurs processeurs respectifs, assurant ainsi une correspondance exacte et une simulation rapide. De plus, des options de déverminage et d'analyse, compatibles avec les autres outils de CoWare, y ont étés ajoutées. Plusieurs architectures de bus sont présentes dans la bibliothèque et peuvent être représentées au niveau transactionnel ou RTL selon le besoin. De plus, il est possible d'y connecter en même temps des composants décrits à ces deux niveaux d'abstraction grâce à des interfaces automatiquement adaptables. Les bus sont munis d'une bonne capacité d'analyse, permettant d'identifier les goulots d'étranglement, et offrent un support particulier pour les architectures AMBA et OCP.

2) **Processor Designer.** Ce deuxième outil de CoWare permet de créer des processeurs sur mesure pour une application donnée. La conception de processeurs est réalisée en utilisant LISA 2.0 (*Language for Instruction Set Architecture*) qui est, comme son nom l'indique, un langage permettant de décrire avec formalisme et précision l'architecture et le jeu d'instruction d'un processeur [52]. Le langage LISA, dont la syntaxe s'inspire fortement du langage C, vise à couvrir plusieurs classes de processeurs (e.g. RISC, DSP, VLIW). Un des objectifs principaux du langage LISA était de permettre la modélisation au cycle près d'un processeur, ce qui n'est pas chose facile étant donné la présence d'un pipeline complexe pour certaines classes de processeurs. À partir de la description du processeur en langage LISA, l'outil Processor Designer génère automatiquement une série d'outils et de modèles

[1]Jusqu'à récemment, ces outils étaient nommés respectivement ConvergenSC Model Libraries, LISATek Processor Designer et ConvergenSC Platform Architect

permettant son utilisation:

- un compilateur optimisateur pour le langage C, un éditeur de lien et un archiveur permettant de générer un exécutable pour l'architecture cible,
- des outils de profilage et d'analyse intégrés permettant d'optimiser le logiciel pour l'architecture donnée avec une emphase sur la puissance, la surface ou la vitesse du processeur,
- un ISS pouvant être intégré dans une plateforme SystemC, telle Platform Architect de la même compagnie, ou autres types de plateforme,
- une description VHDL ou Verilog synthétisable du processeur permettant de l'implémenter dans un produit final.

3) **Platform Architect.** Ce logiciel de CoWare favorise la conception et l'intégration de SoC complet. Il offre principalement un environnement de développement et de vérification basé sur Eclipse, nommé SCIDE (*SystemC Integrated Development and Debug Environment*), pour la réalisation de modules décrits en SystemC ou importés d'autres outils de la suite comme Model Libraries, Processor Designer ou Signal Processing Designer (non décrit ici). L'environnement graphique offre la possibilité de facilement assembler des blocs ensemble, mais l'intérêt principal de Plateform Architect est au niveau de la vérification et du déverminage. Premièrement, le déverminer est adapté aux constructions de SystemC comme l'exécution sur plusieurs processus. Deuxièmement, l'outil permet de générer un graphique en forme d'onde des différents signaux et horloges du système. Troisièmement, il permet de tracer l'ordre des événements, processus et appels aux fonctions des différentes interfaces (IMC, *Interface Method Call*) sous forme graphique. Un autre avantage de SCIDE est que son compilateur permet d'optimiser les changements de contextes dus aux processus, signaux et ports de SystemC, accélérant ainsi la simulation.

1.1.2.3 Roses

La plateforme Roses, issue du groupe de recherche TIMA à Grenoble, vise à générer automatiquement les interfaces de communication dans un MPSoC hétérogène. La conception de ces interfaces spécialisées nécessite normalement une couche de OS décrite à bas niveau qui est entremêlée avec l'application. La programmation de cette couche est généralement fastidieuse et présente un goulot d'étranglement pour le développement des systèmes embarqués. L'objectif de Roses est de générer automatiquement cette couche et de fournir un API qui permet de lui accéder facilement. L'approche utilisée par Roses est de type ascendante ou encore basée sur les composants [18]. Ce type d'approche démarre avec un ensemble de composants et vise à générer les API de communication. Par opposition, une approche descendante démarre avec une architecture fonctionnelle, généralement décrite à un haut niveau d'abstraction, et raffine vers le bas pour obtenir une implémentation RTL de la plateforme. SPACE aborde ce deuxième type d'approche. La génération d'interfaces de Roses procure trois avantages important [17] :

1. le logiciel dédié n'a pas à se soucier de l'interface vers le matériel;
2. l'application peut être développée indépendamment de l'implémentation matérielle; et,
3. offre un niveau de flexibilité et de portabilité accru sans perte de performance.

Un système sur Roses est défini en utilisant SystemC auquel on a étendu certaines fonctionnalités. Roses utilise une architecture virtuelle composée de modules virtuels, de ports virtuels et de canaux virtuels. Un module virtuel représente un module de la plateforme accompagné d'une interface constituée de plusieurs ports virtuels. Un port virtuel quant à lui est composé de plusieurs ports internes et externes aux composants. Celui-ci permet de relier les ports internes et externes en convertissant

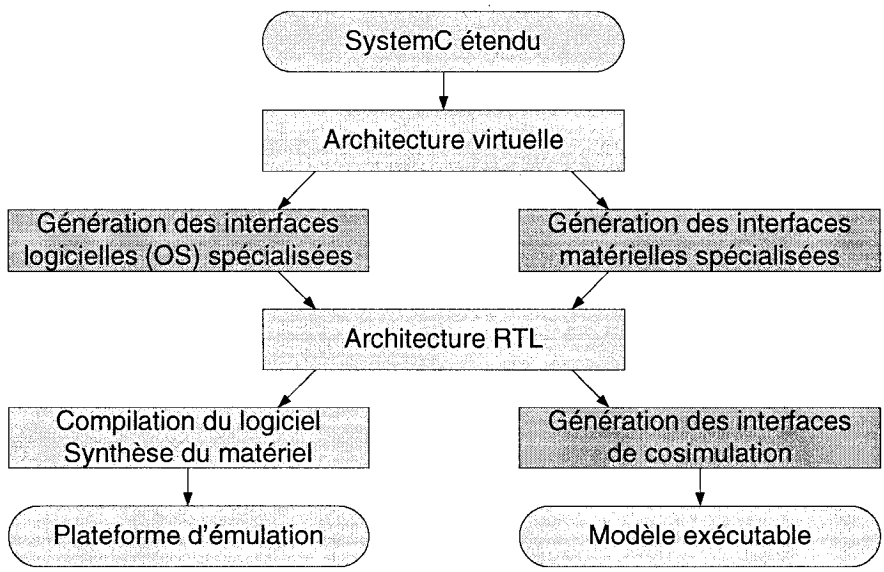

Figure 1.2 Environnement de Roses

le protocole ou le niveau d'abstraction lorsque nécessaire. Finalement, les canaux virtuels permettent de relier ensemble les ports virtuels tout en masquant la plupart des détails de communication. De plus, pour des fins de raffinement, le modèle est annoté à l'aide de paramètres architecturaux, comme par exemple le protocole utilisé.

La figure 1.2 illustre l'environnement de Roses et les trois types d'interfaces automatiquement générées. Les interfaces matérielles sont générées à partir d'une bibliothèque de services préconfigurée pour chaque processeur ciblé. Un avantage est qu'un graphe de dépendance permet de n'inclure que les services nécessaires, réduisant au minimum la taille de l'OS. Les interfaces matérielles sont générées à partir d'une bibliothèque contenant des modèles de processeurs et de protocoles pour les canaux. Finalement, le générateur de cosimulation produit un modèle exécutable SystemC. De plus celui-ci agit également à titre de pilote pour d'autres simulateurs (e.g. VHDL, Verilog, ISS) grâce à des adapteurs.

Bien que Roses soit un outil puissant pour interfacer ensemble des composants d'un système hétérogène, il n'offre pas de support pour l'exploration architecturale et le

partitionnement. Pour combler cette lacune, une approche a été proposée combinant Roses et VCC (*Virtual Component Co-Design*) pour obtenir un flot de conception complet permettant d'obtenir une implémentation RTL à partir des spécifications du système [26]. VCC est une autre plateforme de développement qui se concentre sur l'exploration architecturale [56], c'est donc une approche descendante telle que décrit précédemment. VCC fourni des outils permettant d'associer des blocs fonctionnels à des composants architecturaux, créant un partitionnement logiciel / matériel. Par la suite, les performances du système sont analysées et on peut joué avec la configuration de l'architecture pour obtenir un partitionnement idéal. On obtient donc à la fin un modèle abstrait du système. C'est là que Roses entre en jeu, pour procurer en lien direct et automatique du modèle abstrait de VCC vers une plateforme Roses. Une fois le système converti en Roses, il devient possible de générer les interfaces des modules et de raffiner le système vers une implémentation RTL.

1.1.2.4 Autres Plateformes

Il existe plusieurs autres plateformes de développement à haut niveau. Parmi ces autres plateformes, notons le projet MESCAL (*Modern Embedded Systems, Compilers, Architectures and Languages*) qui vise à fournir une approche formelle au développement de SoC [43]. Leurs outils visent à trouver une correspondance entre la concurrence présente dans une application et la concurrence présente dans la plateforme. L'exploitation de la concurrence se fait à quatre niveaux [7]:

1. Au niveau bit via des unités fonctionnelles spécialisées
2. Au niveau instruction grâce à des processeurs VLIW
3. Au niveau processus légers par des processeurs multiprocessus
4. Au niveau processus par l'utilisation de plusieurs processeurs

L'utilisateur spécifie d'abord une architecture pour le système. Des outils de compilation et de simulation pour la partie logicielle sont alors automatiquement

générés. Le compilateur de MESCAL analyse ensuite l'application et permet d'exploiter le parallélisme en fonction de l'architecture choisie. Finalement, du code est généré pour chaque élément de calcul (PE, *processing element*) qui inclue l'ordonnancement et la synchronisation.

TTL (*Task Transaction Level*) est une autre de ces plateformes de développement pour les MPSoC [61]. Leur approche est basée sur la définition d'un système par un ensemble de tâches et par les interfaces entre ces tâches. TTL supporte plusieurs types de communication en offrant une grande variété de types d'interfaces (e.g. bloquant ou non-bloquant, requêtes ordonnées ou non ordonnées). La fonction de multi-tâches de TTL est réalisée en offrant trois types de tâches :

1. Processus qui correspond à une tâche n'ayant pas d'interaction explicite avec d'autres tâches. Celles-ci n'interagissent pas directement avec l'ordonnanceur, bien qu'elles peuvent être suspendues par la plateforme, comme par exemple un OS.
2. Co-routine qui sont des tâches exécutant en collaboration avec d'autres tâches et qui interagissent avec l'ordonnanceur du système.
3. Acteur qui représente une tâche ayant un début et une fin. Autrement dit, celle-ci correspond plus à une fonction qu'à une tâche, un peu comme les `sc_method` de SystemC.

ArchiFlex (*Architecture Flexible*) présente une solution pour l'exploration architecturale différente de celles présentées précédemment. Cette plateforme n'en est pas une de simulation mais d'émulation, c'est à dire qu'elle permet d'avoir une implémentation physique de la plateforme et non pas un modèle. Comme il a été expliqué, l'élévation du niveau d'abstraction permet d'obtenir des vitesses de simulation accrues, mais celles-ci ne se comparent pas à la vitesse d'exécution du circuit réel, c'est donc là l'avantage principal d'une plateforme d'émulation. L'autre avantage d'ArchiFlex est que la plateforme procure une grande flexibilité au niveau

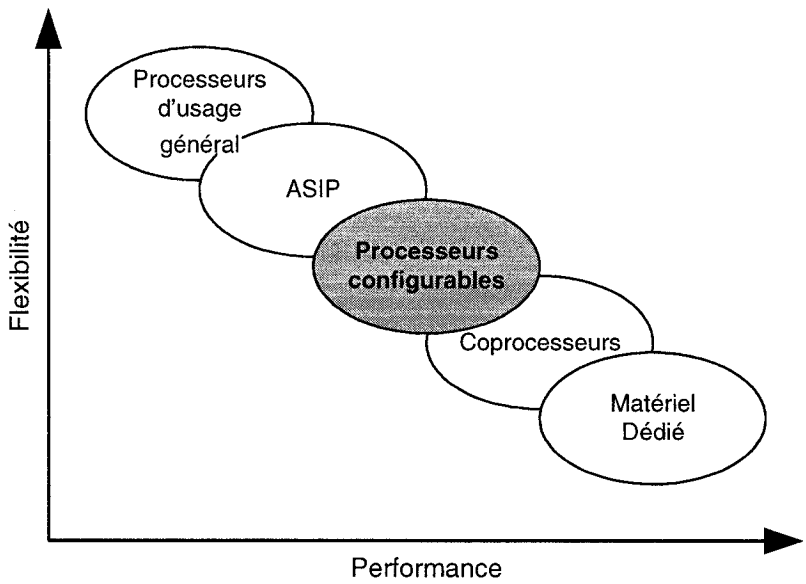

Figure 1.3 Position des processeurs configurables

des choix architecturaux, principalement grâce à un réseau sur puce reconfigurable et des interfaces de communication génériques, contrairement à ses compétiteurs dont la plateforme est centrée sur leurs propres produits [45].

Finalement, parmi les plateformes de développement, notons la présence de la plateforme StepNP qui sera présentée en détail à la section 3.1 puisqu'elle fait l'objet de ce travail.

1.2 Processeurs configurables

Pour accélérer la vitesse d'exécution de l'application tout en maintenant un bon niveau de flexibilité, une approche ayant fait son apparition récemment est l'utilisation de processeurs configurables. Cette classe de processeurs, présentée à la figure 1.3 [25], constitue un compromis entre la flexibilité offerte par les processeurs dont le jeu d'instruction est dédié à une classe d'applications (ASIP, *Application Specific Instructions Set Processor*) et la performance offerte par les coprocesseurs dédiés.

Bien que les processeurs configurables ne rivalisent pas en puissance de calcul brute avec les coprocesseurs dédiés, ceux-ci offrent un avantage important au niveau de la facilité de conception et d'intégration, permettant de réduire considérablement le temps de mise en marché. De plus, la flexibilité qu'ils offrent permet de mieux s'adapter aux changements fréquents de spécifications (e.g. nouveaux standards, mises à jour mineures), réduisant davantage le temps de mise en marché et augmentant la réutilisation des composants [22]. Les processeurs configurables offrent principalement des extensions à trois niveaux architecturaux [34] :

Extension du jeu d'instruction : Les processeurs configurables offrent la possibilité d'étendre le jeu d'instruction (ISA, *Instruction Set Architecture*) du processeur en y ajoutant des instructions spécialisées, développées sur mesure pour une certaine classe d'applications. L'intégration des nouvelles instructions varie d'un processeur à l'autre. Dans certains cas, celles-ci sont directement des nouvelles instructions assembleur et sont exécutées par le pipeline du processeur. Dans d'autres cas, un bloc de logique est ajouté à côté du processeur pour exécuter ces instructions et celui-ci est contrôlé par des instructions du processeur.

Sélection de blocs prédéfinis : Lors de la configuration du processeur, le concepteur à la possibilité d'ajouter ou de retirer certaines unités (e.g. multiplicateur entier, unité de traitements en points flottants, registres spéciaux) pouvant contribuer à l'accélération de l'application ou à d'autres facteurs désirables.

Paramétrage : Les processeurs configurables offrent la possibilité de fixer certains paramètres architecturaux comme par exemple la taille des caches, le nombre de registres ou la largeur du bus. Ces paramètres permettent de réaliser de bons compromis entre la performance et le coût (e.g. puissance, surface, fréquence d'horloge), dépendamment des contraintes fixées.

Les processeurs générés par l'outil Processor Designer de CoWare à l'aide du langage LISA (section 1.1.2.2) sont un premier exemple de processeur configurable puisqu'ils offrent toutes les caractéristiques décrites précédemment. De plus, puisque ces processeurs sont décrits très précisément à l'aide d'un langage spécialisé, les options de configuration sont extrêmement flexibles. Évidemment ceci peut également encourir une configuration plus complexe pour un usager qui voudrait tirer profit au maximum des options de LISA.

1.2.1 Xtensa

La compagnie Tensilica, avec son processeur vedette Xtensa, s'est illustrée comme chef de file dans le domaine des processeurs configurables. Nous présentons ici deux versions de leur processeur, soit une version plus ancienne, le Xtensa T1050, et une version plus récente, le Xtensa LX. Comme cette dernière version n'était pas disponible au début de cette recherche, la version T1050 fut utilisée.

1.2.1.1 Xtensa T1050

Bien que ce processeur ne dispose pas du niveau de flexibilité offert par LISA, les options de configurations disponibles aux concepteurs sont tout de même nombreuses [32]. Le Xtensa possède une configuration de base que partagent toutes les configurations possibles. Celle-ci comporte un jeu d'instructions de 80 instructions et un pipeline cinq étages s'inspirant tous deux d'une architecture RISC standard. Une caractéristique intéressante du Xtensa pour les systèmes embarqués est qu'il permet de réduire la taille d'un programme en utilisant des instructions de 16 et 24 bits. De plus, comme tous les processeurs de systèmes embarqués, le Xtensa est conçu pour avoir une faible consommation de puissance. À cette configuration de base, il est possible d'ajouter ou de modifier certaines options via une interface conviviale et

facile d'utilisation disponible directement sur le site web de Tensilica. Quelques unes des options de configuration sont présentées ci-dessous :

Blocs prédéfinis	Paramètres architecturaux
Multiplicateur entier	Taille et adresse des différentes mémoires externes
Fonctionnalités de DSP (nommé Vectra)	
Unité de traitement en point flottant	Ordre des octets en mémoire
Unité de gestion de la mémoire	Largeur du bus externe
Registre booléen	Taille et type des caches

Le Xtensa supporte également l'ajout d'instructions spécialisées au jeu d'instruction de base. Les instructions sont définies dans un langage nommé TIE (*Tensilica Instruction Extension*) qui s'inspire de la syntaxe de Verilog. Le langage TIE et ses différentes caractéristiques seront étudiés plus en détail à la section 2.2.

1.2.1.2 Xtensa LX

Avec son processeur Xtensa LX, Tensilica apporte principalement deux importantes améliorations à ses versions précédentes. Dans un premier temps, on ajoute le compilateur XPRES (*Xtensa Processor Extension Synthesis*) qui permet de déterminer automatiquement les extensions à apporter au processeur. Dans un deuxième temps l'ajout des ports et queues TIE qui fournissent une interface de communication rapide entre les processeurs.

XPRES: Le compilateur XPRES exécute et profile l'application étudiée en utilisant une configuration de base du Xtensa. Par la suite, il ajoute des unités de traitements permettant d'accélérer l'application selon les résultats trouvés. Puisque le compilateur XPRES démarre avec une solution fonctionnelle et qu'il ajoute des unités connues, ceci permet une optimisation plus simple et plus rapide que d'utiliser des outils générant un

processeur à partir de zéro (e.g. LISA). Un grand avantage de XPRES est qu'il explore un grand éventail de solutions et présente au développeur un graphique illustrant les diverses accélérations atteignables et le coût, en surface supplémentaire, qui leur est associé. Ainsi, le développeur peut sélectionner le compromis vitesse / surface qui répond le mieux aux contraintes du système. XPRES utilise trois techniques permettant de réaliser ces accélérations soit [39]:

1) La fusion d'instructions. La fusion d'instructions consiste à repérer à l'intérieur des boucles de contrôle les séquences d'instructions qui sont fréquemment répétées, par exemple plusieurs décalages suivis d'un " ou " binaire. Le compilateur XPRES tentera par la suite de combiner ces instructions en une seule instruction TIE. Le gain réalisé est double puisque le temps d'exécution des opérations est réduit et le nombre d'instructions à lire de la mémoire est également réduit.

2) Les instructions SIMD (*Single Instruction Multiple Data*). Il est fréquent que les boucles de contrôle effectuent la même opération (e.g. addition, multiplication) sur plusieurs données d'un vecteur. Dans ce cas, XPRES peut ajouter une instruction TIE qui réalise cette opération sur plusieurs données à la fois. Lors de son exploration, XPRES donne les accélérations pour 2, 4 et 8 unités de calcul respectivement, donnant ainsi plus de flexibilité en termes du compromis entre l'accélération et la surface.

3) Les instructions FLIX (*Flexible-Length Instructions Extensions*). Les instructions FLIX du Xtensa LX sont très similaires à une architecture VLIW (*Very Long Instruction Word*). Un problème des architectures VLIW, particulièrement du point de vue des systèmes embarqués, est qu'elles entraînent une augmentation substantielle de la taille du code [57]. Ceci vient du fait que la taille des instructions est fixe et que très souvent le compilateur n'arrive pas à remplir complètement le code d'instruction étant donné les dépendances. Ceci entraîne qu'une bonne partie des codes d'instruction est inutilisée, c'est à dire remplacée par des instructions NOP (*No*

Operation).

Contrairement aux instructions VLIW classiques, les instructions FLIX ne sont pas formées de la simple concaténation du code original des instructions à exécuter. Lors de la conception d'une instruction FLIX, qui se fait de manière similaire à une instruction TIE, le nombre de fentes disponibles ainsi que les instructions qui peuvent y être placées doivent être spécifiés. Par exemple, une nouvelle instruction FLIX pourrait être créée en spécifiant que celle-ci contient deux fentes pour les instructions, que la première fente peut contenir les instructions ADD2X, SUB et SUB2X et que la deuxième fente peut contenir les instructions BNE et S8I[2]. De plus, l'instruction NOP est automatiquement ajoutée à la liste des instructions potentielles pour une fente puisqu'elle peut être nécessaire dans le cas où le compilateur ne réussirait pas à remplir toutes les fentes. L'instruction FLIX décrite précédemment permettrait donc d'exécuter douze combinaisons de deux instructions différentes, soit quatre pour la première et trois pour la seconde.

Une fois l'instruction décrite, le compilateur TIE détermine le nombre de bits nécessaires pour chacune des fentes en fonction des instructions qui peuvent y être placées et il génère la logique nécessaire pour décoder et exécuter la nouvelle instruction. Les instructions FLIX peuvent avoir une taille de 32 ou 64 bits et le nombre de fentes disponibles pour les instructions peut être compris entre deux et quinze. Par contre, plus le nombre de fentes est grand, plus les instructions disponibles doivent être spécifiques et en petit nombre. Une instruction FLIX 64 bits de quinze fentes ne laisse que quatre bits pour encoder chaque instruction. Un autre avantage des instructions FLIX est qu'elles peuvent contenir des instructions TIE, obtenues par exemple par fusion ou extension SIMD, procurant un deuxième niveau de parallélisme. De plus, ces instructions peuvent être entremêlées avec des instructions de base de 16 ou 24 bits, permettant de garder le code plus compact aux endroits où l'application

[2]ADD2X, SUB, SUB2X, BNE, S8I et NOP sont des instructions de base du Xtensa.

ne nécessite pas d'accélération.

Les instructions FLIX ne sont pas une exclusivité de XPRES, ces instructions peuvent être spécifiées explicitement si le développeur le désire. Par contre, XPRES détermine automatiquement les bons candidats pour ces instructions et les utilise aux endroits appropriés, ce qui représente une tâche laborieuse à effectuer manuellement.

Interfaces multiprocesseurs: Les techniques classiques de communication d'une plateforme MPSoC incluent généralement l'utilisation d'une zone de mémoire commune pour transférer des données, soit par un mécanisme de passage de messages ou encore de multitraitement symétrique (SMP, *Symmetric Multiprocessing*). Plusieurs techniques d'optimisations existent pour gérer ces échanges de données. Par contre, elles comportent toutes le désavantages que les données transigent sur un canal commun, entraînant un goulot d'étranglement au niveau des communications. Pour résoudre ce problème, Tensilica a ajouté au Xtensa LX deux mécanismes de connexion entre les processeurs, les ports et les queues.

Les ports, illustrés à la figure 1.4 [39], fournissent des interfaces directes de communication entre les processeurs. Ces ports peuvent servir d'opérande de sortie ou d'entrée aux instructions spécialisées du processeur. De plus, dès qu'un processeur écrit une donnée sur le port elle devient immédiatement disponible pour celui qui la lit, procurant une interface très rapide. Un avantage intéressant des ports de communication du Xtensa est que leur taille est indépendante de la taille du bus externe et qu'elle n'est pas limitée à des puissances de deux. Un port de 40 bits, par exemple, serait tout à fait acceptable.

Les queues, figure 1.5 [39], fonctionnent de manière similaire aux ports décrits précédemment. Comme dans le cas des ports, les queues peuvent être utilisées comme opérande d'entrée ou de sortie dans les instructions spécialisées. Les opérations de lecture et d'écrire dans la queue peuvent être bloquantes ou non, dépendamment

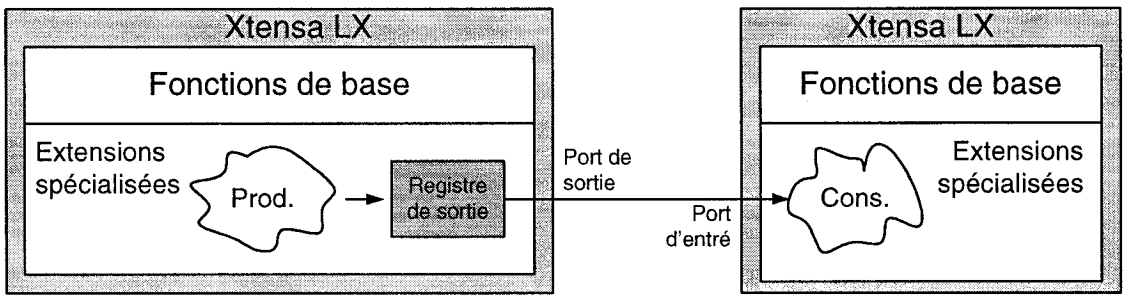

Figure 1.4 Schéma d'un port TIE

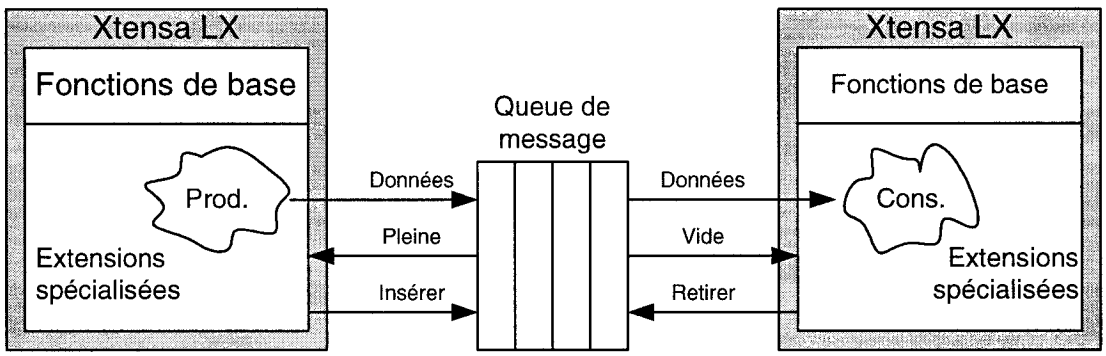

Figure 1.5 Schéma d'une queue TIE

des besoins de l'instruction. L'utilisation des opérations bloquantes donne l'avantage de desservir les processeurs le plus rapidement possible. Par contre, les opérations non bloquantes permettent au processeur d'exécuter un autre processus en attendant que la queue soit disponible. L'établissement de la liaison (*handshaking*) entre les processeurs et les queues est géré automatiquement par l'interface, facilitant leur utilisation. De plus, l'interface simple du Xtensa le rend facilement adaptable à une queue standard [59]. La largeur et profondeur des queues peut être fixée indépendamment de la largeur de l'interface mémoire standard et ne nécessite pas d'être une puissance de deux. De plus, les processeurs peuvent avoir plusieurs queues d'entrée et de sortie.

Vectra LX : Vectra est une option de configuration du Xtensa visant les applications de traitement de signaux. Ce module ajoute à la configuration de base une large

quantité d'instructions SIMD, seize registres 160 bits et une deuxième unité de chargement de 128 bits. Le Vectra LX, combiné avec les instructions spécialisée, procure de solides performances tout en ayant une très faible consommation de puissance [11].

1.2.2 Autres processeurs configurables

1.2.2.1 Stretch

La série de processeurs S5000 de Stretch est basée sur la combinaison d'un processeur RISC standard et d'une unité programmable d'instructions spécialisées (ISEF, *Instruction Set Extension Fabric*). Le processeur RISC utilisé est un Xtensa version T1050, configuré avec les options de MAC (*Multiply And Accumulate*), d'unité de calcul en virgule flottante et un bus de 128 bits [6]. Bien que le Xtensa soit lui-même un processeur configurable, dans le S5000 sa configuration est fixe et on ne l'utilise que comme RISC de base. La raison pour laquelle les développeurs ont choisi ce processeur peut probablement s'expliquer par le fait qu'une partie de l'équipe de Stretch a travaillé à la conception du Xtensa, donc possède une bonne expérience avec ce processeur.

Le ISEF est responsable d'exécuter les instructions spécialisées définies par le concepteur. À la différence des instructions TIE du Xtensa, le ISEF est une unité externe au pipeline. Le ISEF possède 32 registres de 128 bits chacun qui peuvent être utilisés comme opérandes pour les instructions spécialisées. Lors de l'exécution d'une instruction spécialisée, les opérandes sont lues à partir des registres généraux du Xtensa ou encore des registres du ISEF dépendamment du cas. Les arguments sont passés au ISEF durant les étages trois (exécution) et quatre (mémoire) du pipeline. Au cinquième étage du pipeline, le ISEF commence sont exécution de l'instruction.

L'exécution d'une instruction spécialisée peut être réalisée sur plusieurs cycles, jusqu'à un maximum de 27.

Un autre élément qui distingue les processeurs S5000 des processeurs Xtensa est que la syntaxe utilisée pour spécifier les instructions spécialisées est fortement inspirée du C. Ceci permet de garder le même langage pour les instructions spécialisées que pour l'application elle-même, en supposant que l'application soit décrite en C.

Une autre caractéristique importante des processeurs de Stretch est qu'ils sont reconfigurables, contrairement aux processeurs Xtensa qui sont simplement configurables. Un processeur S5000 peut posséder plusieurs configurations de ISEF qui contiennent chacune un certain nombre d'instructions spécialisées. En cours d'exécution, le processeur peut changer la configuration du ISEF suite à une directive explicite du programme ou encore si le programme tente d'exécuter une instruction spécialisée ne se trouvant pas dans la configuration courante. Cette propriété permet de réaliser des économies sur la surface utilisée par le processeur.

1.2.2.2 Garp

L'architecture Garp comporte plusieurs similitudes avec l'architecture S5000 de Stretch. Tout d'abord, il s'agit d'un processeur général, un MIPS dans ce cas ci, combiné à un coprocesseur reconfigurable fortement couplé au processeur [15]. Lors de la compilation, le compilateur C de Garp identifie toutes les boucles et les transforme en diagramme de flot de données. Pour accélérer l'exécution et augmenter le parallélisme, le coprocesseur pipeline les itérations de la boucle à exécuter tout en respectant les dépendances entre les itérations. Une autre technique utilisée pour accélérer l'application est l'exécution spéculative des chargements.

Une fois que les instructions à exécuter sur le coprocesseur sont identifiées, le

compilateur effectue l'assignation des fonctions vers les unités fonctionnelles du coprocesseur. Le processeur MIPS a le contrôle du coprocesseur reconfigurable et dispose de fonctions permettant le transfert des données entre les deux unités. Puisque les transferts de données ne se produisent qu'à l'entrée et la sortie des boucles, on suppose que la surcharge causée par ce transfert est petite. Le processeur et le coprocesseur peuvent exécuter indépendamment l'un de l'autre. Chaque boucle à accélérer correspond à une configuration du coprocesseur et le coprocesseur ne peut avoir qu'une seule configuration à la fois. Lorsque le coprocesseur nécessite une nouvelle configuration, il la télécharge de la mémoire et se modifie en conséquence. Une mémoire cache pour les configurations récentes permet d'accélérer ce processus et la reconfiguration est relativement rapide puisque celles-ci sont de petites tailles.

1.2.2.3 APU

L'APU (*Auxiliary Processing Unit*) de Xilinx est une solution utilisant un FPGA (*Field Programmable Gate Array*) pour accélérer certaines fonctionnalités du processeur [5]. Pour ce faire, Xilinx utilise un Virtex-4 et un processeur PowerPC. Le schéma de la figure 1.6 illustre le concept de ce processeur. Lors d'une nouvelle instruction, le code correspondant est décodé simultanément par le pipeline de base et par le contrôleur de l'APU. S'il s'agit d'une instruction spécialisée, elle sera exécutée dans le bloc de logique spécialisée du FPGA et le résultat sera retourné au pipeline à la fin de l'étage d'exécution, mais pas nécessairement au même cycle d'horloge. Les instructions exécutées par l'APU peuvent être synchronisées soit de manière à ce que le processeur poursuive son exécution pendant que le coprocesseur fait ses calculs ou encore qu'il bloque en attendant que l'instruction soit complétée. En plus d'ajouter la possibilité de créer des instructions spécialisées, l'APU de Xilinx fournit une extension points flottants au PowerPC.

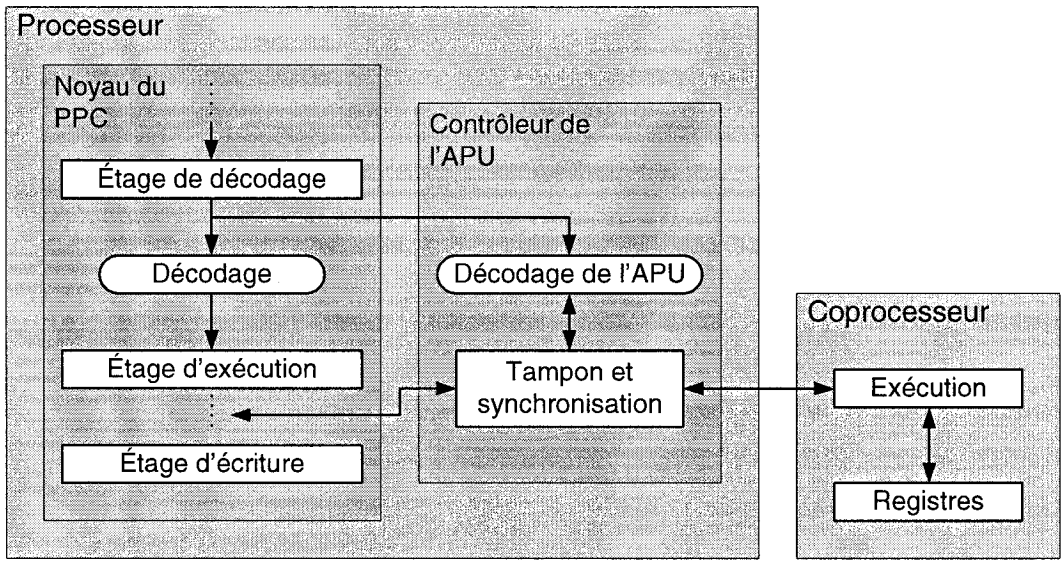

Figure 1.6 APU de Xilinx

1.2.2.4 Triton-Builder

Cet outil, de la compagnie Poseidon, permet de générer automatiquement des accélérateurs matériels pour une grande variété d'applications [2]. L'outil de développement profile l'application, simulée avec la plateforme matérielle du système, et identifie les points critiques au niveau des boucles et des fonctions. Comme le fait le Xtensa LX, l'outil Triton-Builder propose à l'utilisateur une série d'accélérations potentielles et leur performance, ce qui permet de faire un choix éclairé sur les points à optimiser. Par la suite, l'outil génère un coprocesseur au niveau RTL, en VHDL ou Verilog, et les interfaces nécessaires pour y accéder. Un avantage de Triton-Builder par rapport aux autres solutions décrites précédemment est qu'il n'est pas lié à un processeur particulier, il peut être utilisé soit avec un ARM, un PowerPC, un Microblaze ou encore un Nios.

CHAPITRE 2

EXTENSION DU JEU D'INSTRUCTIONS POUR UN ENCODEUR MPEG-4

Ce chapitre présente en premier lieu la méthodologie de conception utilisée dans cette recherche. On y présente également en détails la première étape de cette méthodologie qui consiste à accélérer une application en ajoutant des instructions spécialisées. Finalement, on y décrit l'application de cette première étape à un encodage MPEG-4.

2.1 Méthodologie

2.1.1 Méthodologie de Quinn et Lavigueur

Pour établir notre méthodologie nous nous sommes basé sur la méthodologie proposée par Quinn et Lavigueur [53] qui est une extension de la méthodologie classique de codesign. Cette méthodologie, présentée à la figure 2.1, comporte principalement les quatre grandes étapes décrites ci-dessous :

1. Démarrer avec une solution purement logicielle. Cette étape permet de se doter, premièrement, d'une spécification formelle et exécutable. Ensuite, elle vise à maximiser l'utilisation du logiciel pour éviter l'utilisation de matériel dédié. Finalement, cette étape inclut également une vérification du code de base pour s'assurer qu'aucune fonction n'est mal codée et utilise inutilement un grand nombre de cycles du processeur.

2. Choisir une configuration adéquate. Dans l'approche de codesign classique, il

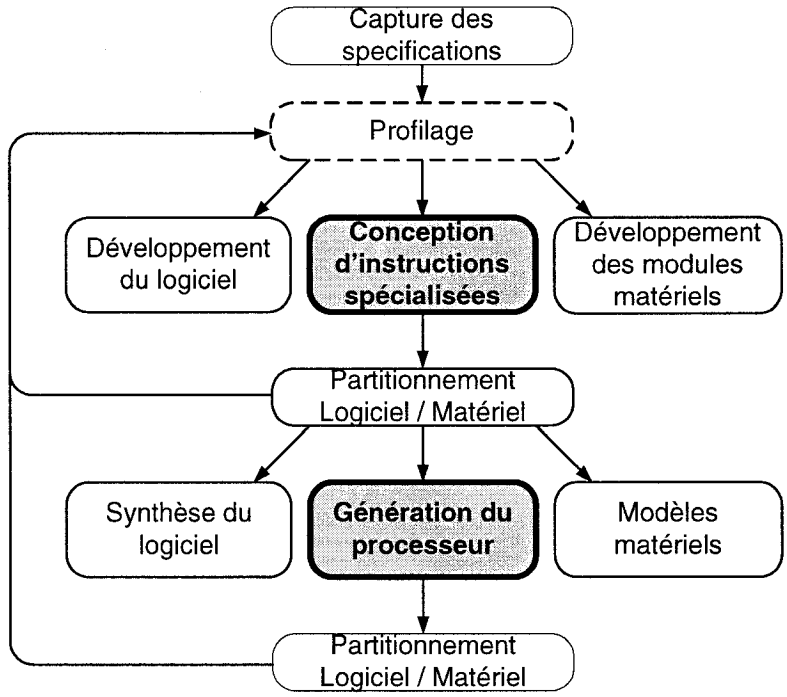

Figure 2.1 Méthodologie de codesign de Quinn et Lavigueur

était nécessaire de sélectionner la classe de processeurs appropriée pour notre application (e.g. DSP, RISC standard, processeurs réseaux). Aujourd'hui, avec l'utilisation des processeurs configurables, ceci peut se traduire par la sélection des options de configuration appropriées pour notre application (e.g. MAC, taille des caches, nombre de registres).

3. Ajout d'instructions spécialisées. L'utilisation d'instructions spécialisées permet d'obtenir des gains en performance fort appréciables. De plus, leur conception et intégration se fait généralement aisément, tout particulièrement si on utilise des outils automatisés comme le compilateur XPRES de Tensilica (voir section 1.2.1.2).

4. Ajout de coprocesseurs. Bien que les instructions spécialisées soient très utiles, elles ont tout de même leurs limites, par exemple au niveau des accès mémoires. Advenant le cas où les instructions spécialisées ne permettent pas d'atteindre les

accélérations voulues, il devient nécessaire d'ajouter des coprocesseurs dédiés.

2.1.2 Méthodologie de Tensilica

Tensilica propose également une méthodologie pour la conception d'instructions spécialisées qui est décrite à la figure 2.2 [58]. Cette méthodologie s'applique principalement à la version T1050 du Xtensa puisque la version LX détermine automatiquement les instructions spécialisées. Lors de cette recherche, la version T1050 est celle qui fut utilisée puisque la version LX n'était pas disponible.

La méthodologie débute par l'identification des points critiques, opération qui peut souvent être faite par simple inspection du code, mais pour facilité cette opération Tensilica fournit le logiciel xt-gprof, un outil de profilage basé sur gprof de GNU. Cet outil indique entre autres le nombre d'appels de chaque fonction ainsi que le nombre de cycles d'horloge passés dans chacune d'elles, permettant de mieux identifier les sections critiques de l'application.

À chaque itération, le développeur peut utiliser les instructions TIE pour accélérer un certain point critique de l'application. Ceci peut ce faire soit en améliorant les instructions TIE déjà existantes ou en accélérant une section qui n'a pas encore été modifiée.

La restructuration du code, étape précédant la réalisation d'instructions TIE, vise à placer les points critiques de l'application à l'intérieur de fonctions. De cette manière il devient facile de remplacer la version originale par la version TIE et vice-versa, permettant ainsi de vérifier le bon fonctionnement et d'explorer plus aisément diverses solutions.

Une fois les instructions TIE décrites, les outils de Tensilica génèrent automatiquement les outils de développement (e.g. compilateur, éditeur de lien, bibliothèques,

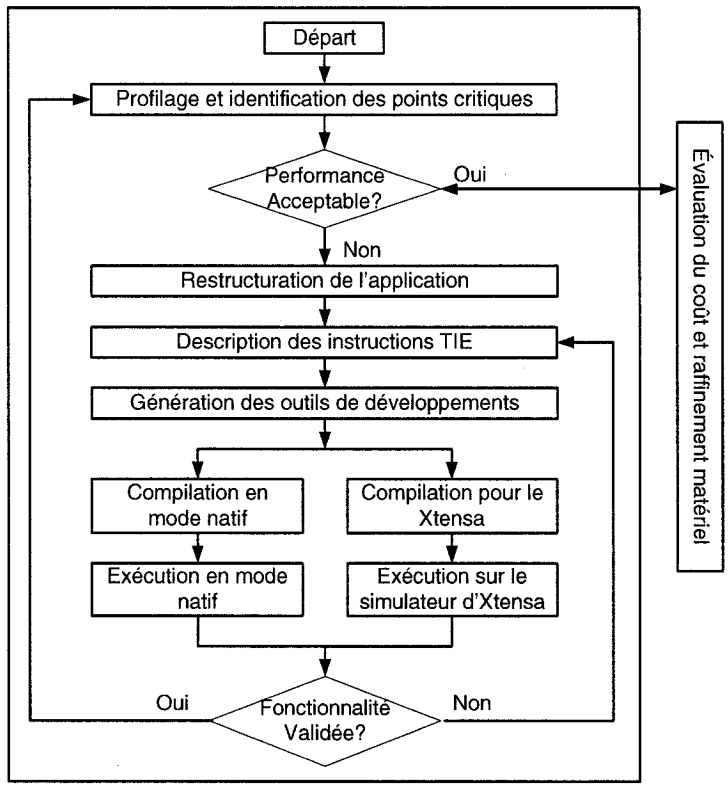

Figure 2.2 Méthodologie de Tensilica

ISS) qui permettront de simuler l'application à laquelle les instructions spécialisées ont été ajoutées. Tensilica fournit deux environnements permettant la simulation de l'application. La première option est d'exécuter l'application sur l'ISS du Xtensa. Ceci permet d'obtenir des résultats de profilage qui seront utiles pour déterminer l'efficacité des nouvelles instructions et évaluer les points critiques qui mériteront d'être accélérés lors des itérations subséquentes. Deuxièmement, il est possible de compiler l'application en mode natif, dans ce cas les outils de développement remplacent les instructions TIE par un équivalent en C. Cette deuxième option ne donne pas de résultats de profilage, par contre elle permet de valider rapidement la fonctionnalité des instructions spécialisées puisque ce mode de simulation est plus rapide que le mode ISS par plusieurs ordres de grandeur.

Une fois la performance satisfaisante, la phase suivante consiste à évaluer le coût

matériel de la nouvelle instruction. Les outils de Tensilica peuvent générer le code VHDL correspondant à l'instruction, par la suite un outil de synthèse standard (e.g. Design Analyser) peut donner la taille du circuit correspondant pour une technologie donnée. La double flèche entre les étapes « Performance Acceptable? » et « Évaluation du coût et raffinement matériel » indique qu'il est possible de revenir au design des instructions TIE si le coût de ces instructions dépasse un niveau acceptable.

2.1.3 Méthodologie utilisée

La méthodologie qui est présentée ici concerne la phase d'exploration architecturale. Le flot de conception est constitué principalement de trois étapes successives soit: l'accélération de l'application à l'aide d'instructions spécialisées, la distribution du calcul sur plusieurs processeurs et finalement l'ajout de coprocesseurs dédiés. L'ajout principal à cette méthodologie concerne la parallélisation multiprocesseur qui procure une possibilité supplémentaire pour l'accélération avant l'utilisation de matériel dédié.

La méthodologie décrite à la figure 2.3 démarre avec la capture des spécifications, étape où seront spécifiés les requis en terme de vitesse. Pour l'instant, nous nous sommes penché principalement sur la vitesse d'exécution de l'application. Des améliorations futures pourront considérer en plus la puissance consommée et la surface requise. La première itération considère une solution entièrement logicielle s'exécutant sur un seul processeur sans aucun type d'accélération. La satisfaction des contraintes suivant cette approche est favorisée puisque c'est la meilleure au niveau du temps de développement, de la flexibilité et du coût.

La méthodologie proposée est composée de trois phases qui permettent d'accélérer notre application. Chaque itération démarre avec le profilage de l'application et se termine avec la simulation et la validation du système. Ce processus se continue jusqu'à ce que les contraintes soient atteintes. La première phase consiste à utiliser

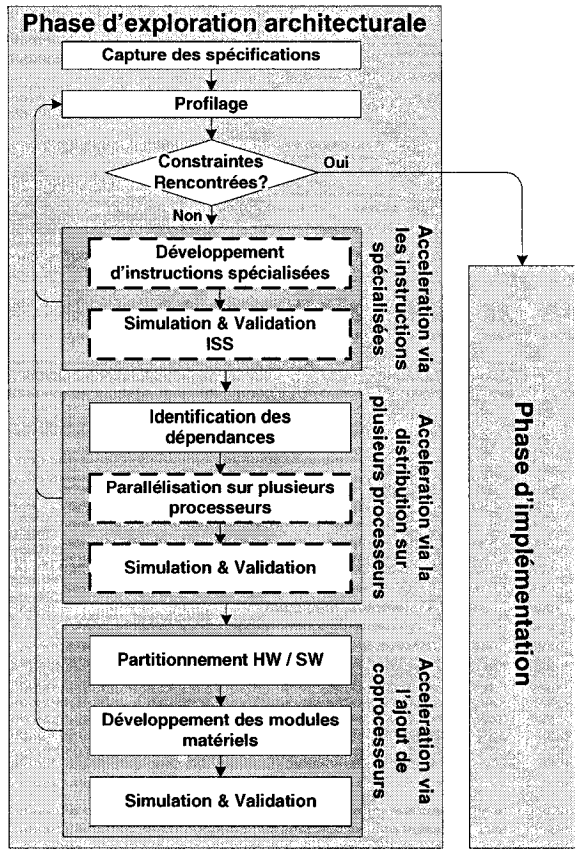

Figure 2.3 Méthodologie utilisée

les processeurs configurables pour étendre le jeu d'instruction du processeur. Il a été montré que cette technique permet de réaliser des gains en vitesse importants avec un surplus de matériel relativement petit et un temps de développement court [53]. Pour ces raisons, ce type d'accélération sera préféré aux autres techniques. Dans le cas d'un projet développé sur le Xtensa, comme c'est le cas ici, Tensilica fourni tous les outils nécessaires à la conception des instructions spécialisées, avec son langage TIE et son compilateur, ainsi qu'à la simulation et validation, avec son ISS. La suite de ce chapitre décrit cette étape et présente sa mise en oeuvre sur une application d'encodage MPEG-4.

Advenant le cas où les instructions spécialisées ne permettraient pas d'atteindre des performances satisfaisantes, l'étape suivante consiste à paralléliser l'application sur

plusieurs processeurs. Par soucis de simplicité, nous nous sommes concentré sur la distribution sur plusieurs processeurs configurables homogènes. Par contre, comme discuté à la section 4.3, la distribution sur plusieurs processeurs hétérogènes possède un grand potentiel et fait l'objet d'améliorations possibles. Pour les applications qui sont orientées donnés, comme par exemple dans les domaines de traitement d'images et de réseautique, l'ajout de processeurs permet d'atteindre un gain en performance pratiquement linéaire avec le nombre de processeurs, à condition que la contention sur le canal de communication soit faible. Les instructions spécialisées ne permettent pas d'accélérer les accès mémoire. Par contre, comme il sera montré à la section 2.4.1, elles peuvent contribuer à en réduire le nombre, permettant ainsi une meilleure extension lorsque le nombre de processeurs augmente en réduisant la contention. Puisque les instructions spécialisées ont un potentiel d'accélération beaucoup plus grand que linéaire, la parallélisation sur plusieurs processeurs ne sera utilisée qu'en deuxième phase. Le développement et la simulation d'un système multiprocesseur est plus complexe qu'un système monoprocesseur. Pour nous aider dans cette tâche, nous utilisons la plateforme StepNP, qui sera décrite au chapitre 3. Les résultats de simulation de cette deuxième phase seront donnés au chapitre 4.

La troisième étape de notre méthodologie, utilisée si les instructions spécialisées et la parallélisation multiprocesseur ne parviennent pas à obtenir les gains voulus, est d'ajouter des coprocesseurs dédiés. Cette dernière méthode procure le plus haut niveau de performance, mais au prix de la flexibilité, du coût et du délai de conception. Le fait de retrouver le partitionnement logiciel / matériel si loin dans le développement vient de notre objectif de privilégier l'utilisation du logiciel. Puisqu' aucun accélérateur matériel n'est présent aux étapes un et deux, il n'y a pas lieu de décider d'un partitionnement avant l'étape 3.

Lorsque les contraintes de performance sont atteintes, la phase d'exploration architecturale est complétée et nous passons à la phase d'implémentation. Cette phase

comporte principalement les étapes de raffinement et synthèse du matériel, création du processeur spécialisé et intégration du système.

2.2 Instructions TIE

Le langage TIE, tel que mentionné à la section 1.2.1, fut développé afin de permettre la description des instructions spécialisées. Sa principale force est la facilité avec laquelle ces nouvelles instructions sont intégrées. Nous décrirons ici certaines des caractéristiques fournies par le langage TIE qui peuvent être combinées entre elles pour former une description détaillée des nouvelles instructions. Cette description comporte deux parties principales soit l'énumération des opérandes impliquées dans l'instruction ainsi que la description de la fonctionnalité réalisée.

Il existe plusieurs types d'opérandes disponibles pour les instructions TIE. Premièrement, on retrouve, comme pour toutes instructions standard, le fichier de registre de base et les valeurs immédiates. Le fichier de registre possède deux ports de lecture et un d'écriture. Les instructions spécialisées, comme les instructions de base, sont donc limitées à deux lectures et une écriture sur celui-ci. La plage des valeurs immédiates valides pour une instruction peut être restreinte au gré du concepteur. Ceci possède les avantages de minimiser la logique nécessaire pour implémenter l'instruction et réduit la partie du code d'instruction nécessaire pour inscrire la valeur. Une autre manière disponible pour spécifier une valeur immédiate est l'utilisation d'une table de constantes. Celle-ci étant accédée par indice spécifié par une des opérandes de l'instruction.

Le langage TIE permet d'ajouter des fichiers de registres supplémentaires au processeur, en plus de celui de base, pouvant également servir d'opérandes aux instructions spécialisées. Les outils du Xtensa génèrent automatiquement toute

logique nécessaire au contrôle des interruptions, exceptions et aléas du pipeline reliés à ce nouveau fichier de registres. De plus, ceux-ci ne sont pas limités aux mêmes contraintes que celui de base. Ainsi, les nouveaux fichiers de registres peuvent avoir soit 2, 4, 8, 16 ou 32 registres et la largeur de ceux-ci peut aller jusqu'à 1024 bits et n'a pas à être une puissance de deux. Les fichiers de registres spécialisés n'ont aucune limite sur le nombre de ports de lecture, le nombre exact de ports nécessaires étant déterminé par le compilateur, par contre ils sont tout de même soumis à la limite d'un port d'écriture. Pour pouvoir utiliser ces nouveaux fichiers de registres, le compilateur génère de nouveaux types de variables associés à ce fichier de registres utilisables dans un programme C, ainsi que des fonctions de chargement et stockage. Outre les nouveaux fichiers de registres, les instructions spécialisées peuvent ajouter des registres spéciaux, ou états (*state*) en langage TIE, qui permettent de garder des valeurs en mémoire. Les registres spéciaux ont essentiellement la même fonctionnalité qu'un fichier de registres à un seul registre. L'efficacité de leur utilisation est montrée dans [42] pour garder les valeurs fréquemment accédées dans les boucles.

Au niveau de la description de la fonctionnalité d'une instruction TIE, celle-ci ce fait par une suite d'opérations et d'assignations. Le tableau 2.1 illustre la majorité des opérations disponibles en langage TIE. Une fonctionnalité intéressante du langage est qu'il permet de définir des instructions multi-cycles, permettant d'implémenter de longues chaînes d'opérations. Le compilateur du Xtensa est en mesure d'insérer correctement des bascules pour séparer ce type d'instruction, par contre il est recommandé d'utiliser des outils de synthèse plus spécialisés (e.g. Design Compiler de Synopsys) pour obtenir une parfaite balance entre les différents étages. Un problème relié au délai des instructions est que la seule manière de savoir si une instruction est trop longue est de faire la synthèse. L'inconvénient est que l'outil qui la fait ne voit pas les instructions mais seulement une suite d'opérations logiques. Donc le chemin critique est spécifié sous forme d'une suite de registres et de portes logiques, ce qui rend l'identification de l'instruction ayant causé la faute de délai difficile.

Tableau 2.1 Opérations disponibles dans le langage TIE

Type d'opérations	Opérations	Symboles
Arithmétiques	addition, soustraction, multiplication	`* + -`
Logiques	et, ou, négation	`&& || !`
Relationnelles	plus grand, plus petit, égal	`> < ==`
Binaires	et, ou, ou exclusif	`& | ^`
Décalages	à droite, à gauche	`» «`
Concaténations	concaténation	`{}`
Conditionnelles	assignation conditionnelle	`a = (cond) ? b : c`

Un autre problème est que, par défaut, chaque instruction est accompagnée de sa propre logique et il n'y a aucun partage de ressources. Toutefois, il est possible de décrire d'une manière plus précise l'implémentation matérielle d'une instruction et de spécifier le partage de ressources qui existe entre plusieurs instructions, résultant en une implémentation plus compacte des instructions spécialisées. Malgré le fait que les opérations disponibles permettent de réaliser des instructions très complexes, il est important de toujours garder à l'esprit que ces instructions ont un coût matériel qui leur est associé. Ainsi, une instruction TIE multipliant ensemble 100 nombres en un seul cycle aura besoin de 99 multiplicateurs différents, dépassant possiblement la surface disponible pour le processeur.

2.3 Application d'encodage MPEG-4

L'encodage MPEG-4 est une opération qui vise à compresser un vidéo pour en réduire la taille utilisée en mémoire. La taille d'un vidéo non compressé étant gigantesque, une seconde de vidéo non compressé pris à partir d'un téléphone cellulaire pouvant facilement occuper une douzaine de mégaoctets, cette opération est nécessaire pour pouvoir stocker ou transmettre un vidéo sans surcharger les ressources requises. Au départ, l'image est divisée en petits morceaux nommés macroblocs. Chaque

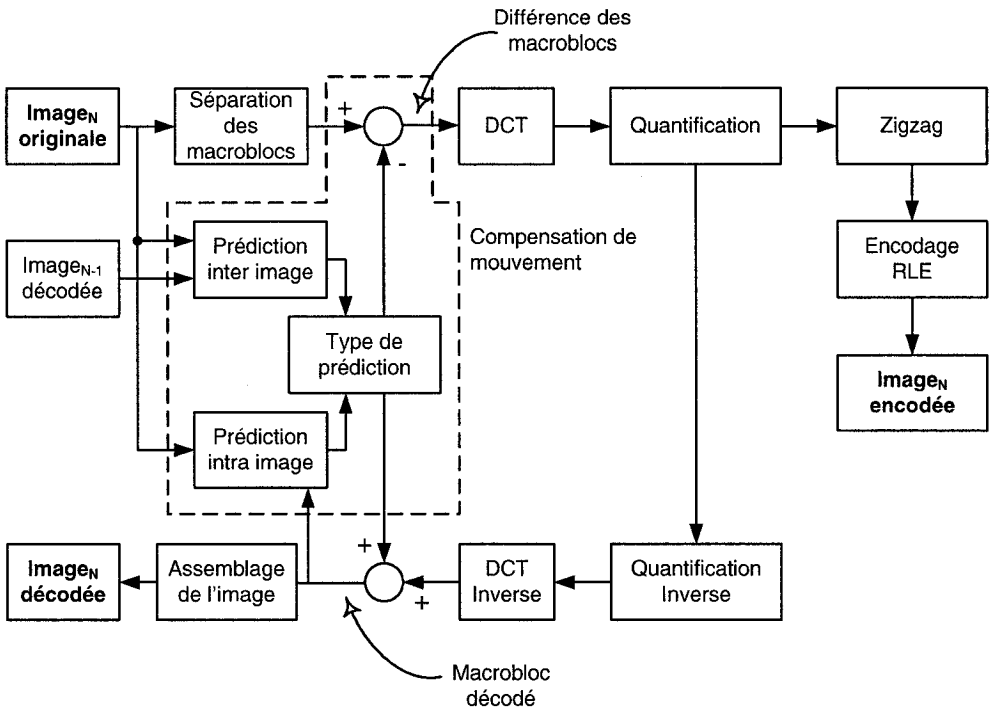

Figure 2.4 Encodage MPEG-4

macrobloc contient quatre sous blocs de 8x8 pixels représentant la luminance (niveau de gris) et deux, quatre ou huit blocs de 8x8 pixels représentant la chrominance (niveaux des couleurs). Généralement, seulement deux blocs de chrominance sont utilisés car l'oeil distingue mieux l'intensité de la lumière que le niveau des couleurs. Une fois l'image divisée, une suite d'opérations, illustrée à la figure 2.4 [54], est effectuée sur les macroblocs.

Compensation de mouvement. Un des principes derrière l'encodage vidéo est que les macroblocs sont fortement corrélés dans le temps et l'espace. C'est à dire, il y a de fortes chances pour qu'un macrobloc soit très semblable à ses voisins dans la même image et à ceux situés environ au même endroit dans l'image précédente. Afin de tirer profit de cette différence, lors de l'encodage d'un macrobloc on cherche en premier lieu un macrobloc ayant une forte similitude avec celui à encoder. Par la suite on effectue la différence entre les deux macroblocs et on encode cette différence

plutôt que d'encoder directement le macrobloc. La prédiction de mouvement peut se faire de deux manières différentes. La prédiction inter image est basée sur la corrélation temporelle, donc le macrobloc est cherché dans l'image précédente. Cette première technique est généralement la plus efficace et sera par conséquent employée pour la majorité des images. La prédiction intra image, quand à elle, est basée sur la corrélation spatiale, donc le macrobloc est comparé à ses voisins dans la même image. La prédiction intra est également utilisée pour l'encodage d'image (e.g. JPEG). Dans les deux cas, le macrobloc est toujours comparé aux macroblocs de l'image décodée et non aux macroblocs de l'image originale. Ceci vient du fait que la compression MPEG-4 entraine une perte d'information. Autrement dit, l'image décodée ne sera pas identique à l'image originale. Puisque le décodeur ne dispose pas de l'image originale, l'encodeur se doit de faire la différence de macroblocs avec l'image telle qu'elle sera vue par le décodeur. Ce qui résulte de l'opération de compensation de mouvement est donc un macrobloc représentant la différence entre le macrobloc à encoder et un macrobloc de référence.

DCT. La transformée en cosinus discrète (DCT, *discrete cosine transform*) est une opération qui transforme un bloc de 8x8 dans une représentation du domaine fréquentiel. Le principal avantage de cette opération est que les fréquences varient peu à l'intérieur du macrobloc, particulièrement à cause de l'opération de compensation de mouvement qui laisse un macrobloc majoritairement vide. De plus, cette opération permet de rassembler les basses fréquences dans le coin supérieur gauche du bloc. À elle seule, cette opération n'ajoute rien à la compression puisque ce n'est qu'une représentation différente de la même information. Elle constitue par contre une étape nécessaire pour tirer avantage des étapes suivantes.

Quantification. Cette étape divise les valeurs du macrobloc par un certain facteur de quantification entier et tronque le résultat. Cette troncation est à l'origine de la perte d'information dans l'encodage vidéo et explique pourquoi l'image décodée ne sera pas

parfaitement identique à l'image originale. Par contre, cette perte d'information se produit au niveau des changements de fréquence, autrement dit le contour des formes, et l'oeil humain est mal adapté pour voir cette différence. Un avantage important de l'opération de quantification est que la région supérieure gauche de chaque blocs du macrobloc, dont les valeurs sont très petites suite à la DCT, sera arrondie à zéro, limitant la quantitié d'information à encoder.

Zigzag. Comme il a été mentionné, la zone inférieure droite des blocs du macrobloc contient des valeurs de zéro suite à l'opération de quantification. L'opération de zigzag consiste à réordonner les pixels de chaque bloc selon le patron illustré à la figure 2.5. Cette dernière opération a pour effet d'aligner un maximum de zéros à la fin du vecteur, permettant de tirer profit au maximum de l'opération suivante.

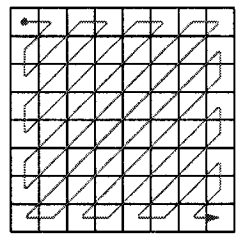

Figure 2.5 Zigzag

Encodage RLE. Avec un encodage RLE (*Run Length Encoding*, plutôt que d'indiquer les valeurs des pixels un après l'autre, on indique la valeur des x prochains pixels (e.g. les 17 prochaines valeurs sont zéro). Puisque les étapes précédentes ont produit un vecteur contenant une grande suite de zéros, l'application de l'encodage RLE permet d'atteindre un haut taux de compression.

Il existe plusieurs variations de l'encodage MPEG-4. La version qui sera utilisée dans notre cas est un algorithme développé par STMicroelectronics pour leur propre usage. L'application vise une vitesse d'encodage de trente images par seconde sur un vidéo possédant une résolution VGA, c'est à dire 640x480 pixels et seize couleurs.

Tableau 2.2 Exploration de la cache

Taille	Associativité	Cycles (milliers)
32k	ensemble de 2 blocs	11 958
16k	ensemble de 2 blocs	11 986
8k	ensemble de 2 blocs	12 033
4k	ensemble de 2 blocs	12 116
32k	directe	12 246

2.3.1 Profilage de l'application et configuration du processeur

Le profilage de l'application a été réalisé sur une seconde de vidéo, soit l'encodage de trente image, et le vidéo en question présente une scène typique de Times Square à New York en plein jour. Cette scène comporte plusieurs personnes et voitures se déplaçant à des vitesses variables ainsi qu'un très grand amalgame de formes et de couleurs. Une première passe de profilage révèle que la fonction de multiplication entière est responsable à elle seule de plus de 40% du temps d'exécution du processeur. L'ajout d'une unité de multiplication à notre configuration de base semble donc de mise et permettra de faire passer le nombre de cycles nécessaires de 22 millions à 12 millions.

Un second point intéressant à observer est le nombre d'accès mémoires. Comme toute application orientée données, ce nombre est élevé et représente dans ce cas ci environ 30% des instructions exécutées par le processeur. L'utilisation d'un bus de 128 bits semble donc approprié pour notre application. Malheureusement, le Xtensa n'est pas conçu pour prendre automatiquement avantage de cet ajout, dû probablement au fait que l'architecture demeure 32 bits, puisque l'accélération obtenue suite à l'extension du bus à 128 bits n'est que de 1.02. Par contre cette extension sera tout de même maintenue car elle permet de réaliser des instructions TIE intéressantes, comme il sera montré à la section 2.4.1

Un troisième point qui a été examiné est la taille et le type de mémoire cache

utilisée. Le tableau 2.2 présente les résultats de plusieurs tests effectués en ce sens. On remarque que la cache n'a pas une influence marquante sur la performance de l'application. Ceci peut s'expliquer par le fait que l'encodage est fait un macrobloc à la fois. Ainsi, puisque la majorité des calculs se font sur un ensemble très petit de données, celui-ci tient entièrement dans la cache. Lorsque l'encodage d'un macrobloc est terminé, il ne sert plus, exception faite de l'opération de compensation de mouvement qui réutilise les macroblocs passés. Ceci permet donc d'expliquer la faible différence entre les divers types et taille de cache. Une analyse de la cache après réalisation des instructions spécialisées a produit des variations du même ordre de grandeur.

Finalement, l'ajout de l'extension vectra DSP aurait pu produire des résultats d'accélération très intéressants. Par contre, considérant sa très grande taille, il n'a pas été utilisé.

Après avoir effectué ces diverses modifications au processeur, on obtient le profilage illustré à la figure 2.6 qui constitue notre point de départ pour le développement des instructions spécialisées.

2.4 Conception d'instructions spécialisées pour le MPEG-4

Cette section présente les diverses instructions spécialisées qui ont été développées pour accélérer l'application d'encodage MPEG-4 ainsi que les résultats obtenus. Il s'agit là des résultats de la phase un de la méthodologie exposée à la section 2.1.3.

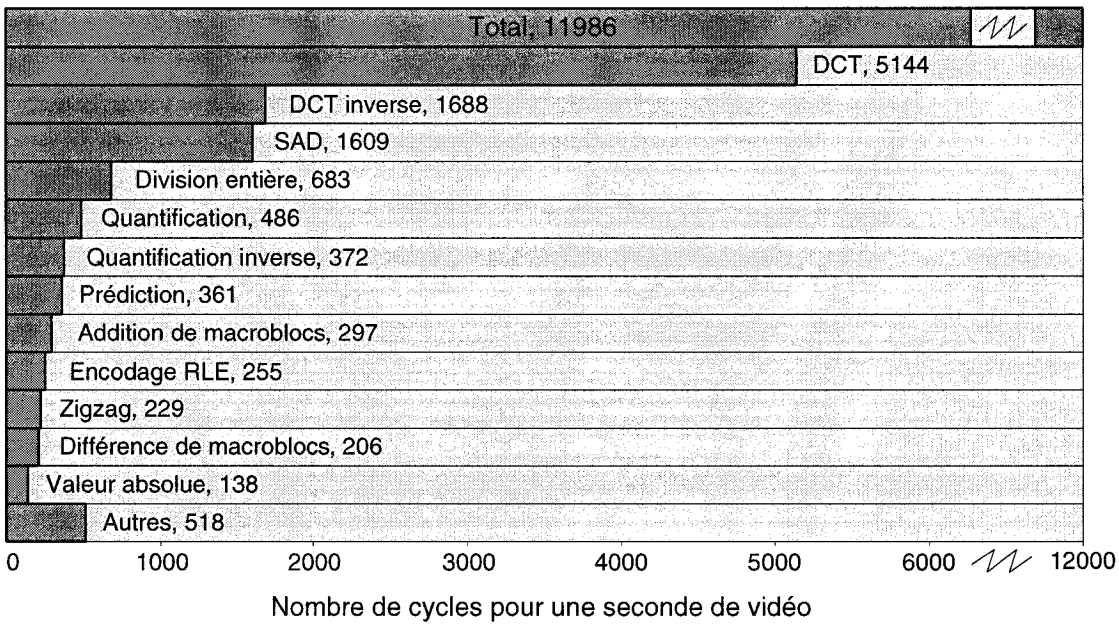

Figure 2.6 Profilage de l'application MPEG-4

2.4.1 Détails des instructions spécialisées

2.4.1.1 DCT

La fonction qui calcul la DCT sur un bloc 8x8 est réalisée en utilisant une multiplication matricielle entre le bloc et des coefficients fixes. Le produit réalisé est illustré ci-dessous.

$$\begin{bmatrix} coeff0\_0 & coeff0\_1 & \dots & coeff0\_63 \\ coeff1\_0 & coeff1\_1 & \dots & coeff1\_63 \\ \dots & \dots & \dots & \dots \\ coeff63\_0 & coeff63\_1 & \dots & coeff63\_63 \end{bmatrix} \times \begin{bmatrix} src0 \\ scr1 \\ \dots \\ src63 \end{bmatrix} = \begin{bmatrix} dst0 \\ dst1 \\ \dots \\ dst63 \end{bmatrix}$$

src = bloc 8x8 d'entrée coeff = coefficients fixes dst = bloc 8x8 destination

Chacune des 64 données du bloc est lue 64 fois à l'intérieur d'une boucle de contrôle. Ces données ne peuvent être entièrement contenues dans les registres généraux du processeur, donc il y a beaucoup de transferts entre la cache et les registres. La

première optimisation réalisée vise à réduire le nombre de ces transferts. En utilisant les registres spéciaux du Xtensa (*state*), il est possible de garder en mémoire les 64 données du bloc et de les rendre toutes directement accessibles au processeur. Pour limiter davantage le nombre de transferts de la cache, nous utilisons le bus de 128 bits du Xtensa pour transférer les données, codées sur 32 bits, quatre à la fois. Pour effectuer le lien entre la mémoire et les registres spéciaux, des instructions de chargement et de rangement ont été ajoutées au processeur. De plus, ces instructions incrémentent le pointeur de quatre éléments en même temps qu'elles effectuent l'accès, sauvant ainsi une instruction de plus. De cette manière, il est possible de charger ou d'emmagasiner les 64 registres du processeur en seize instructions comme suit :

```
inline void load_TIE_register(int* ptr) {
  LOAD4_0(ptr);  // Chargement de 4 donnés dans les reg. 0 à 3
  LOAD4_1(ptr);  // Chargement de 4 donnés dans les reg. 4 à 7
  ...
  LOAD4_15(ptr); // Chargement de 4 donnés dans les reg. 60 à 63
};

inline void store_TIE_register(int* ptr) {
  STORE4(ptr,0);  // Emmagisiner 4 donnés à partir des reg. 0 à 3
  STORE4(ptr,1);  // Emmagisiner 4 donnés à partir des reg. 4 à 7
  ...
  STORE4(ptr,15); // Emmagisiner 4 donnés à partir des reg. 60 à 63
};
```

Un deuxième point sur lequel nous pouvons réaliser de bonnes accélérations est au niveau de la parallélisation des opérations. La multiplication matricielle décrite précédemment n'est essentiellement qu'une grande suite d'additions, de soustractions et de multiplications. Le langage TIE nous permet de réaliser des instructions effectuant plusieurs de ces opérations en un seul cycle. Ce comportement émule celui d'une machine SIMD, est sont une manière efficace d'accélérer une application MPEG-4 sans augmenter la puissance consommée [35]. De plus, comme il existe très peu de dépendance entre les données, ces opérations peuvent facilement être réalisées en parallèle. Avec l'ajout de la parallélisation des opérations, l'utilisation des registres spéciaux devient davantage importante. Alors que le fichier de registre de

base possède deux ports de lecture et un d'écriture, chaque registre spécial possède son propre port de lecture et d'écriture. Ceci nous permet donc de lire plusieurs valeurs et d'écrire plusieurs résultats dans la même instruction. Pour le cas de la DCT, puisque la structure du code original s'y prête bien, nous effectuons en une seule instruction quatre groupes de trois additions / soustractions dont le résultat est emmagasiné dans quatre registres spéciaux. Les données utilisées par cette fonction n'utilisent que neuf bits, les pixels d'entrée ont huit bits et les additions de cette fonction augmente ce nombre à neuf. Donc, les additionneurs de cette instruction peuvent n'avoir que neuf bits, réduisant la taille du matériel nécessaire pour implémenter l'instruction. Une seconde instruction permet de multiplier ensemble ces quatre valeurs. Afin de réduire davantage la quantité de calcul à effectuer, les tables servant au calcul des coefficients ont été transférées dans ces instructions spécialisées. Une troisième instruction effectue l'addition de toutes ces valeurs. Ainsi, nous pouvons faire la substitution suivante.

```
  /***** CODE ORIGINAL *****/
// by, byy : pointeurs vers les bloc d'entrée
// u, v, y : compteurs d'itérations
s+= (*(by+0) + *(by+7) + *(byy+0) + *(byy+7)) * DCT_COEFFS[u][v][y][0] +
    (*(by+1) + *(by+6) + *(byy+1) + *(byy+6)) * DCT_COEFFS[u][v][y][1] +
    (*(by+2) + *(by+5) + *(byy+2) + *(byy+5)) * DCT_COEFFS[u][v][y][2] +
    (*(by+3) + *(by+4) + *(byy+3) + *(byy+4)) * DCT_COEFFS[u][v][y][3];

  /***** CODE MODIFIÉ *****/
DCT_ADD_SUB_3(0,0); // Effectue 3 groupes de 4 add/sous
DCT_MUL(index);     // 4 mult. en parallèle
DCT_ADD_ALL();      // Addition des 4 résultats
```

Le code présenté ne reflète pas directement une multiplication matricielle, il a été réorganisé pour tirer profit de la symétrie dans la matrice des coefficients. Finalement, une dernière instruction, montrée ci-dessous, remplace les décalages, opérations logiques et assignations conditionnelles, ce qui constitue également une parallélisation des opérations.

```
  /***** CODE ORIGINAL *****/
// res  : bloc de sortie
```

```
// u, v : compteurs d'itérations
// s    : résultat de la multiplication matricielle
//        pour l'itération u+v*8
res->data[u+v*8] = (((s & 0x80003fff) == 0x80002000) ? (s >> 14) : ((s +
    0x2000) >> 14));

  /***** CODE MODIFIÉ *****/
res->data[u+v*8] = DCT_GET();
```

2.4.1.2 DCT inverse

La fonction de DCT inverse n'est que la multiplication inverse de ce qui a été fait pour la DCT. C'est à dire la matrice des coefficients inversée multipliée par le bloc des valeurs dont on veut obtenir la DCT inverse. Toutefois, l'algorithme est réorganisé pour tirer avantage du fait que la majorité des données d'entrées valent zéro, suite aux opérations de quantification et quantification inverse, ce qui permet d'éviter le calcul de plusieurs multiplications dont le résultat sera également zéro. Cette raison permet d'expliquer la différence entre le temps de calcul de la DCT et de la DCT inverse lors du profilage initial. Pour fins d'optimisation, les mêmes stratégies que pour la DCT ont été utilisées ici. Plus précisément, les registres spéciaux de la DCT sont réutilisés pour garder en mémoire les données des macroblocs, les opérations mathématiques sont parallélisées et les tables statiques sont intégrées aux instructions TIE.

2.4.1.3 SAD

L'opération de SAD (*Sum of Absolute Differences*) prend en entrée deux blocs de 8x8, calcule la différence pixel par pixel, et effectue la somme des valeurs absolues de ces différences. Bref, cette fonction mesure le degré de ressemblance entre deux blocs. Les techniques présentées précédemment, registres spécialisés et parallélisation des opérations, s'appliquent encore dans le cas présent. Un avantage de l'opération de SAD est qu'elle utilise des données codées sur un octet plutôt que sur quatre comme

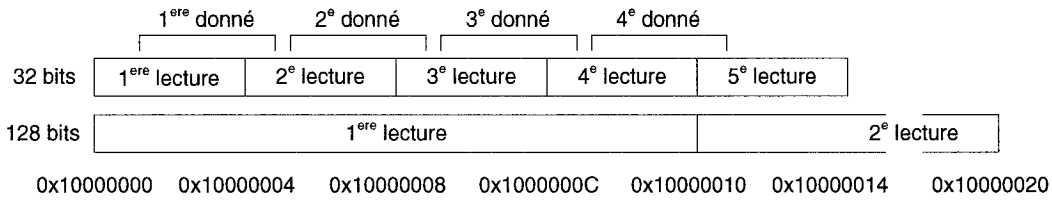

Figure 2.7 Multiples accès non alignés

c'était le cas pour la DCT. Cette propriété de la fonction SAD nous permet donc de lire en un seul accès mémoire quatre ou huit donnés sur un bus de 32 bits ou 64 bits. Par contre, comme les lignes du bloc n'ont que huit octets, il est impossible de tirer pleinement avantage d'un bus de 128 bits. Un problème rencontré, qui ne s'appliquait pas aux autres opérations, est que la lecture de quatre octets à la fois produit des accès non alignés si l'adresse du premier octet n'est pas un multiple de quatre. Normalement, le compilateur du Xtensa prend soin de s'assurer que les entiers soient alignés, mais l'utilisation des instructions spécialisées vient contourner cette règle. Néanmoins, il est possible de gérer les accès non alignés en mémorisant toujours la partie non utilisée de l'accès dans un registre spécial temporaire et en la combinant avec un accès futur. Ainsi, on peut réaliser une suite de lectures non alignées avec seulement une seule lecture supplémentaire. La figure 2.7 illustre la manière de lire quatre données de l'adresse 0x10000001 à l'adresse 0x10000011 en utilisant cinq lectures sur un bus 32 bits ou deux lectures sur un bus 128 bits.

Le code optimisé de la fonction SAD est présenté ci-dessous.

```
  /***** CODE ORIGINAL *****/
// gp : pointeur vers le premier bloc
// rp : pointeur vers le deuxième bloc
// r  : résultat de la somme des différences
for (k = 0; k < siz->x; k++) { // siz->x = 16 toujours
    a = *gp++;
    b = *rp++;
    r += (a > b) ? (a - b) : (b - a); // différence et valeur absolue
}
ok = (r >= lev) ? 0 : 1;
gp += grb->width - siz->x;
rp += ref->width - siz->x;
```

```
  /***** CODE MODIFIÉ *****/
WIEMP_REG(0);
LOAD_UNAL_R1(rp);
for (k = 0; k < 4; k++) {
    LOAD4PIXEL_R0(gp);
    LOAD_UNAL_R1(rp);
    SAD_ADD(3);
}
rp-=4;
```

2.4.1.4 Quantification et quantification inverse

Le coeur de ces deux fonctions est constitué d'une boucle effectuant principalement trois opérations soient : valeur absolue, division entière (quantification seulement) et saturation à douze bits. Trois instructions spécialisées ont été ajoutées pour réaliser chacune de ces opérations. L'opération de valeur absolue se réalise aisément en utilisant un multiplexeur, c'est à dire une assignation conditionnelle, entre la valeur elle-même et sa négation, obtenue par complément à deux. L'opération de division, dont le code source est disponible sur le site web de Tensilica [3], s'obtient en utilisant un algorithme de type décalage et soustraction (*shift and sub*). Bien que la division soit grandement accélérée par l'entremise d'une instruction spécialisée, elle demeure une instruction très lourde et doit être effectuée en cinq cycles d'horloge sur un processeur de 200MHz. Les instructions de valeur absolue et de division sont effectuées sur des valeurs contenues dans des registres d'usage général, ce qui rend ces instructions plus facilement portables. Finalement, l'opération de troncation a été ajoutée pour compléter l'accélération de la quantification. Bien que le code original fasse toujours une troncation à douze bits, par soucis de généralité et de réutilisation, l'instruction TIE de troncation est plus générale et peut faire une troncation à autant de bits que désiré. Suite à ces modifications, nous obtenons le code suivant pour la fonction de quantification.

```
  /***** CODE ORIGINAL *****/
// rp : pointeur vers le bloc d'entrée
```

```
// wp : pointeur vers le bloc destination
// qt : facteur de quantification
// itération sur le nombre de blocs à quantifier
for (i = (pic->mb[a]->mtype & INTER) ? 0 : 1; i < CMAX; i++) {
    val = *rp++;
    lev = abs(val) / (2 * qt); // Division
    val = (val < 0) ? -lev : lev; // Ajustement de signe
    val = (val < -2048) ? -2048 : ((val > 2047) ? 2047 : val); //
        Troncation à 12 bits
    *wp++ = val;
}

  /***** CODE MODIFIÉ *****/
for (i = (pic->mb[a]->mtype & INTER) ? 0 : 1; i < CMAX; i++) {
    val = *rp++;
    UDIV32x16(ABS_TIE(val),(2 * qt)); // Div. et val. absolue en TIE
    lev = (int)Rquotient(); // Récupère résultat de div.
    CLIP_TIE(val,lev); // Troncation en TIE
    *wp++ = val;
}
```

2.4.1.5 Addition et soustraction de blocs

Ces deux fonctions effectuent un calcul très similaire à celui réalisé par la fonction SAD, à l'exception que les résultats des différences / additions ne sont pas additionnés et le résultat est un troisième bloc. Le problème des accès non alignés, par contre, ne se présente pas pour ces fonctions et n'a donc pas à être testé. Le compilateur aligne toujours les images sur un multiple de quatre. De plus, puisque les blocs sont de taille 8x8 et que le calcul se fait un bloc à la fois, ceux-ci demeurent toujours alignés sur une adresse multiple de quatre. La raison pour laquelle ils ne l'étaient pas dans le cas de la fonction SAD est que la recherche des macroblocs voisins s'effectue par déplacements qui ne sont pas multiples de quatre. Le code de la fonction d'addition de bloc a été modifié comme suit et la fonction de soustraction a subit une mutation similaire.

```
  /***** CODE ORIGINAL *****/
// pp : pointeur vers le premier bloc source
// ep : pointeur vers le deuxième bloc source
```

```
// wp : pointeur vers le bloc destination
for (l = 0; l < err->height; l++) {    // Boucle exécutée 8 fois
   for (k = 0; k < err->width; k++) { // Boucle exécutée 8 fois
      x = ((int)(*pp++)) + (*ep++);
      *wp++ = (PIXEL)((x < 0) ? 0 : ((x > 255) ? 255 : x));
   }
   wp += rec->width - err->width;
   pp += prd->width - err->width;
}

  /***** CODE MODIFIÉ *****/
int offset = rec->width;
for (l = 0; l < 8; l++) {
   LOAD_PIXEL0(pp); // pp est un pointeur de " char "
   LOAD4_1(ep);     // ep est un pointeur de " int "
   LOAD4_2(ep);
   BL_ADD_CLIP();   // Addition et troncation
   STORE_PIXEL(wp); // ep est un pointeur de " char "
   wp += offset;
   pp += offset;
}
```

2.4.1.6 Zigzag

La fonction de l'opération de zigzag est extrêmement simple, ce n'est qu'un réordonnancement de valeurs dans un vecteur. Pour accélérer au maximum cette fonction, nous réutilisons encore une fois les registres spéciaux créés pour la fonction DCT et chargeons directement toutes les valeurs du bloc. Par la suite, une instruction spécialisée déplace toutes les valeurs du vecteur pour les réécrire dans le bon registre. L'opération est complétée par une réécriture des registres en mémoire avec les valeurs déplacées. Puisque l'instruction spécialisée n'effectue aucun calcul, la quantité de matériel utilisé pour la réaliser est très faible.

```
  /***** CODE ORIGINAL *****/
for (i = 0; i < 64; i++)
   des->data[zzs[i]] = src->data[i];  // zzs est un tableau contenant
      les indices appropriés pour le déplacement

  /***** CODE MODIFIÉ *****/
load_TIE_register(ptr);
DO_ZIGZAG();
```

```
store_TIE_register(ptr);
```

2.4.1.7 Encodage RLE

La fonction d'encodage RLE prend en entrée un bloc de 8x8 et retourne en sortie deux vecteurs de même longueur indiquant respectivement le nombre de zéro consécutifs et la valeur du pixel suivant cette série. Par example, le bloc d'entrée `[8 0 0 4 0 6 2 0]` retournerait respectivement les vecteurs `[0 2 1 0]` et `[8 4 6 2]`, devant être interprétées comme signifiant : « aucun 0 suivi d'un 8, deux 0 suivis d'un 4, un 0 suivi d'un 6 et aucun 0 suivi d'un 2 ». Connaissant la taille du bloc (8 dans l'example et 64 pour l'application), on peut déterminer le nombre de 0 suivant la dernière valeur non nulle.

Pour cette fonction, nous utiliserons encore une fois la stratégie du parallélisme qui traite quatre données à la fois. Par contre, ici l'opération est plus complexe étant donné qu'il ne s'agit pas d'une opération mathématique répétée sur plusieurs opérandes différentes. Pour accélérer cette opération, une seule instruction a été ajoutée (outre celle de chargement de quatre données qui était déjà présente) et celle-ci effectue la majorité du traitement. À partir des quatre données en entrée, la fonction produit les éléments suivants :

- Écrit dans quatre registres spéciaux les valeurs des données qui ne sont pas égales à zéro. La première valeur non-zéro sera écrite dans le premier registre, la deuxième valeur non-zéro dans le deuxième registre et ainsi de suite, indépendamment de leur position parmi les quatre valeurs initiales.

- Écrit dans quatre autres registres le nombre de zéros qui précèdent la valeur non-zéro correspondante. Dans le cas du premier non-zéro, cette valeur devra être ajoutée aux zéros qui ont étés calculés à l'itération précédente.

Tableau 2.3 Table pour l'encodage RLE

Valeur	Nombre de bits
1^{ere} Valeur différente de zéro	2 bits
2^{e} Valeur différente de zéro	2 bits
3^{e} Valeur différente de zéro	2 bits
Nombre de zéro précédent le 1^{er} non-zéro	2 bits
Nombre de zéro précédent le 2^{e} non-zéro	2 bits
Nombre de zéro précédent le 3^{e} non-zéro	1 bits
Nombre total de non-zéro	3 bits
Nombre de zéro suivant le dernier non-zéro	2 bits

- Écrit dans un registre spécial le nombre de zéros suivant le dernier non-zéro. Cette valeur sera ajoutée à la suite de zéros de l'itération suivante.

- Retourne le nombre de valeurs parmi les quatre qui ne sont pas égales à zéro. Ce nombre sera utilisé dans le code C afin de traiter correctement les valeurs calculées par l'instruction spécialisée.

Pour obtenir ces valeurs, l'instruction spécialisée utilise une table de seize valeurs de seize bits chacune. L'indice permettant d'accéder à cette table s'obtient par un « ou réduit » sur chacune des quatre valeurs d'entrée et en combinant ces quatre bits ensemble. La valeur retournée par cette table est un code permettant d'associer les registres d'entrée aux registres de sortie et donne le nombre de valeurs différentes de zéro parmi les quatre données d'entrées. Les codes utilisés sont résumés dans le tableau 2.3.

Du côté du code C, on charge un bloc de quatre données suivi de l'appel à l'instruction TIE d'encodage. Par la suite, dépendamment de la valeur de retour, qui indique le nombre de valeurs différentes de zéro, on lit les valeurs à ajouter au vecteur résultat directement des registres spécialisés.

```
  /***** CODE ORIGINAL *****/
for (i = 0; i < (src->width * src->height); i++) {  // 64 itérations
   if (src->data[i] != 0) {
```

```
        rl->level[j] = src->data[i];
        rl->run[j] = r;
        r = 0;
        j++;
    } else r++;
}
rl->length = j;

  /***** CODE MODIFIÉ *****/
for(int i = 0; i < 16; i++) {
    LOAD4_0(ptr);
    res = RLE_ENC();
    if(res != 0) {
    switch(res) {
        case 4 : rl->run[j+3] = 0;
                 rl->level[j+3] = RFRAME_SRC03();
        case 3 : rl->run[j+2] = RTEMP_REG2();
                 rl->level[j+2] = RFRAME_SRC02();
        case 2 : rl->run[j+1] = RTEMP_REG1();
                 rl->level[j+1] = RFRAME_SRC01();
        case 1 : rl->run[j] = RTEMP_REG();
                 rl->level[j] = RFRAME_SRC00();
        }
        j+=res;
    }
}
rl->length = j;
```

2.4.2 Résultats obtenus

Les principaux résultats obtenus suite à l'accélération des diverses fonctions par les instructions TIE sont présentés au tableau 2.4. Ce tableau indique, pour chaque fonction donnée dans le profilage initial, le nombre de cycles d'horloge original qu'elle écoulait pour l'encodage d'un vidéo de trente images, le nombre de cycles requis pour ce même encodage suite aux instructions TIE, l'accélération obtenue, la surface supplémentaire nécessaire pour implanter ces instructions et finalement le rapport entre le nombre de cycles économisées et la surface supplémentaire requise. La surface supplémentaire est mesurée par rapport au Xtensa de base qui ne possède aucune instruction spécialisée. La taille requise pour implanter les registres spéciaux, incluant les instructions de rangement et chargement, a été considérée séparément

Tableau 2.4 Résultats des instructions TIE

Fonction	Nombre de cycles (Millions) Original	Avec TIE	Accél.	Augment. de la surface	Accél. par surface
Total	**11986**	**2670**	**4.5**	**68.1%**	**137**
DCT	5144	467	11.0	12.3%	380
DCT inverse	1688	420	4.0	4.0%	317
SAD	1609	366	4.4	2.6%	457
Division entière	683	69	9.9	6.6%	93
Quantification	486	170	2.9	0.4%	790
Quant. inv.	372	124	3.0	0.0%	—
Prédiction	361	361	1.0	—	0
Addition de blocs	297	34	8.7	0.7%	375
Encodage RLE	255	61	4.2	0.5%	388
Zigzag	229	25	9.0	1.9%	107
Différence de blocs	206	27	7.5	0.4%	447
Valeur absolue	138	28	4.9	0.1%	1100
Autres	518	518	1.0	—	0
Registres spéciaux	———	———	——	38.6%	—

étant donné que ces registres sont réutilisés par plusieurs fonctions et que leur taille est très volumineuse.

On observe que la plupart des fonctions parallélisées ont un degré d'accélération égal au niveau de parallélisation, ce qui est le résultat auquel on aurait pu s'attendre. C'est le cas par exemple des fonctions d'addition et soustraction de bloc qui ont un niveau de parallélisation de huit et de la fonction SAD qui a un niveau de parallélisation de quatre. Autre point, l'accélération obtenue pour la fonction DCT est majoritairement due à l'utilisation des registres spéciaux pour garder les valeurs en mémoire. Des tests

ont été réalisés avec seulement la parallélisation des opérations pour cette fonction et les gains n'étaient que de deux, comparativement au gain de onze obtenu avec les registres. Ceci montre bien l'importance de réduire les accès mémoire dans les applications orientées données. Il aurait peut-être été possible de réorganiser le code original pour qu'il effectue moins d'accès mémoires, mais comme cette recherche se concentre sur les instructions spécialisées, cette optimisation ne fut pas tentée. La fonction de DCT inverse n'obtient pas une aussi bonne accélération que la fonction de DCT car le code initial était beaucoup plus efficace, ceci étant dû au fait que cette fonction tire profit des coefficients d'entrées qui valent majoritairement zéro. Finalement, la fonction de prédiction n'a pas pu être accélérée car nous n'avons pas trouvé d'opportunités de parallélisation ou de réduction d'accès mémoire.

Pour ce qui est de la surface supplémentaire requise, on remarque que celle-ci est relativement petite pour la plupart des fonctions. Les opérations réalisées par la majorité des instructions TIE sont des combinaisons d'opérations logiques, d'additionneurs (souvent de petites tailles) et d'assignation conditionnelles. La fonction de quantification inverse ne possède aucune surface supplémentaire étant donné qu'elle réutilise les fonctions de la quantification. Les instructions spécialisées n'ont pas été conçues pour réutiliser les composants, ce qui vient augmenter la surface supplémentaire, particulièrement pour les instructions utilisant des additionneurs. On constate que la fonction DCT, quant à elle, utilise sensiblement plus de matériel que les autres fonctions. Ceci est dû au fait que ses instructions effectuent plus d'additions que les autres instructions, donc nécessite plus d'additionneurs et c'est également la seule fonction qui utilise des multiplicateurs, ceux-ci ayant une taille supérieure à celle des additionneurs. Un point marquant sur la surface additionnelle requise est la taille considérable des registres spécialisés. Le problème est que le Xtensa implémente ces registres en utilisant des bascules D, ce qui est loin d'être efficace en termes de taille et de puissance.

2.4.3 Améliorations possibles

Les instructions TIE réalisées pourraient bénéficier de plusieurs améliorations possibles. Premièrement, l'utilisation de fichiers de registres, implémentés avec des loquets, plutôt que des registres individuels aurait permis de meilleurs résultats de synthèse. Comme mentionné précédemment, l'utilisation d'un fichier de registre possède l'inconvénient qu'il ne permet qu'une seule écriture par instruction, alors que les instructions TIE réalisées effectuent plusieurs écritures. Par contre, outre la fonction de zigzag, le nombre d'écritures réalisées par une instruction TIE est au plus de huit. Donc, une implémentation utilisant huit fichiers de huit registres permettrait de réutiliser les instructions spécialisées, ceci procurerait des gains en performance comparables à ceux présentés à la section précédente tout en diminuant la surface requise.

Un autre point sur lequel il serait possible de sauver de l'espace est la réutilisation des composants. Plusieurs instructions utilisent des additions et chacune d'entre elle possède ses propres additionneurs. Ceci inclut également toutes les instructions de chargement et de rangement puisqu'elles incrémentent l'adresse utilisée. De plus, il est possible de réutiliser les composants réguliers du Xtensa pour diminuer davantage la surface, comme par exemple le multiplicateur présent dans l'unité arithmétique et logique (UAL) qui pourrait servir dans les instructions de la DCT.

Il serait également possible de réaliser des gains de performance plus importants. Lors de la conception des instructions, les fonctions à accélérer ont été considérées séparément. En considérant l'application dans son ensemble et non fonction par fonction, il serait possible de réduire davantage le nombre d'accès mémoire. Par exemple, lors de l'opération de compensation de mouvement, un macrobloc est comparé avec plusieurs macroblocs voisins et chaque comparaison implique l'appel de la fonction SAD. Puisque les macroblocs à comparer se chevauchent, il serait possible

d'un appel de la fonction SAD à un autre de garder dans les registres spéciaux les pixels communs au bloc courrant et au bloc précédent.

La réduction de la taille des variables permettrait également des gains importants en performance. Dans le code utilisé, plusieurs fonctions manipulent des données codées sur quatre octets alors qu'elles n'utilisent en réalité qu'entre neuf à douze bits. En tronquant davantage ces valeurs pour les ramener à un seul octet il deviendrait possible d'en charger et d'en traiter un plus grand nombre en une seule instruction, augmentant le degré de parallélisme. Il va de soi que cette modification ne produirait pas une équivalence parfaite de l'application et que le vidéo encodé perdrait de sa qualité. Par contre, cette perte de qualité serait faible (reste à vérifier si elle demeure acceptable) et elle est potentiellement justifiable si les gains en performance sont suffisamment grands.

CHAPITRE 3

SIMULATION D'UN MPSOC

La deuxième étape de la méthodologie vise à distribuer l'application sur plusieurs processeurs. Pour ce faire, il est nécessaire d'utiliser un outil prévu à cette fin. Les plateformes de développement décrites à la section 1.1.2 pourraient être utilisées ici, mais c'est la plateforme StepNP (*System-Level Exploration of Network Processing Platform*) qui a retenu notre attention. Les facteurs principaux ayant motivé ce choix sont le support pour les plateformes multiprocesseurs et le disponibilité du code source. Le développement de StepNP avait comme objectif initial de:

- Procurer un environnement extensible, modulaire et générique de haute performance pour le développement de plateformes MPSoC
- Fournir une suite d'outils complète permettant d'explorer et d'analyser une plateforme
- Utiliser au maximum les normes de l'industrie et du domaine public

Bien que cette plateforme ait été développée à l'origine pour l'exploration d'applications réseaux [48], ses nombreuses caractéristiques en font un système de choix pour l'exploration de plusieurs autres classes d'applications, comme par exemple les applications multimédia [50], [49]. Les caractéristiques de StepNP ayant motivé son choix sont principalement la disponibilité du code source, facilitant grandement l'intégration de nouveaux processeurs et autres composants (voir section 3.2) ainsi que ses modèles de programmation multiprocesseurs (voir section 3.1.1).

3.1 La plateforme StepNP

Tout comme les plateformes SPACE et Platform Architect, StepNP a été réalisé en utilisant majoritairement le langage C++ et la bibliothèque SystemC. Cette plateforme comporte donc tous les avantages qui y sont rattachés, soit principalement la possibilité de modéliser facilement et rapidement des blocs matériels, l'abstraction entre les communications et leur implémentation via les canaux et la réutilisation de nombreux outils de développement C++.

StepNP est composé de trois environnements de travail interreliés qui forment les blocs de base de la plateforme [48]. Le premier environnement, celui de développement matériel (figure 3.1a), permet de décrire les différents blocs de la plateforme multiprocesseurs. On y retrouve entre autre des modèles de processeurs ARM et PowerPC, une liste de canaux de communication utilisant le protocole SOCP (*SystemC Open Core Protocol*) qui est une version simplifiée de OCP, et des modèles de mémoires. Par la suite, l'environnement de développement logiciel (figure 3.1b) vient préciser l'application qui sera exécutée par les blocs matériels. Finalement, les outils SoC (figure 3.1c) fournissent le nécessaire pour générer, déverminer, contrôler et analyser la plateforme. Les environnements de développement matériel et outils SoC sont décrits plus en détail à l'annexe I alors que l'environnement de développement logiciel sera présenté ci-dessous.

3.1.1 Environnement de développement logiciel

C'est dans cet environnement que les possibilités de StepNP au niveau de la programmation multiprocesseurs prennent forme. En effet, StepNP fournit deux modèles de programmation à haut niveau d'abstraction, sous une suite d'outils nommée MultiFlex, qui permettent de paralléliser une application sur plusieurs

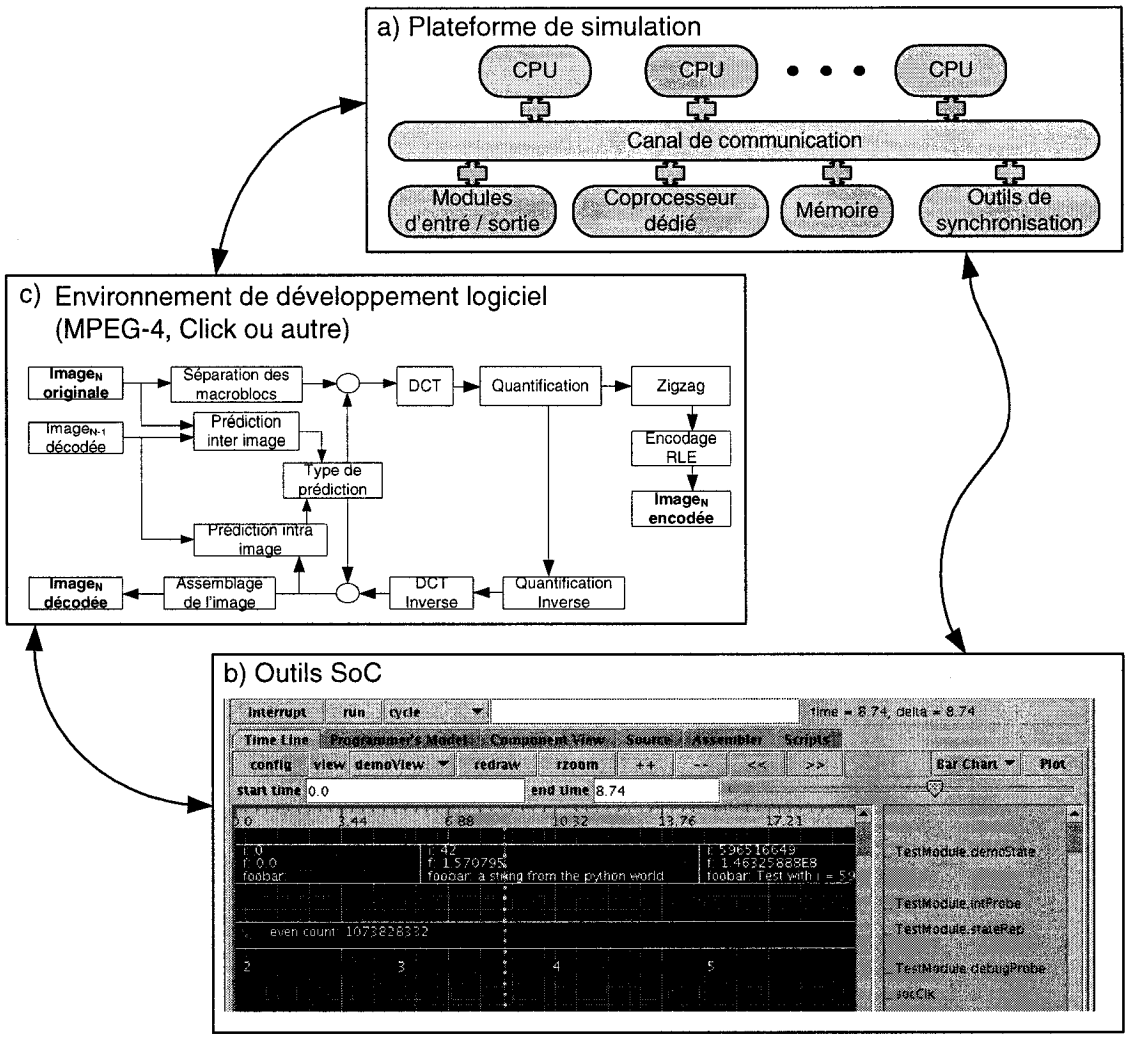

Figure 3.1 Les trois environnements de StepNP

éléments de traitement [50] [51]. Ces deux modèles sont bien différents et sont utilisés dans des contextes également différents, par contre ils sont interopérables, ce qui permet une plus grande souplesse de programmation et une meilleure distribution de l'application. Un point négatif par contre, les outils MultiFlex de StepNP ne fournissent aucun support pour extraire le parallélisme d'une application, la responsabilité de cette tâche repose entièrement sur le programmeur. Ces deux modèles de programmation, nommés DSOC (*Distributed System Object Component*) et SMP (*Symmetrical Multi-Processing*), sont décrits ci-dessous.

3.1.1.1 Modèle DSOC

Le premier modèle de programmation de StepNP, DSOC, concerne la distribution de l'application sur plusieurs éléments de calcul hétérogènes. Pour résumer brièvement le principe, DSOC est une adaptation pour SoC de systèmes distribués comme CORBA ou DCOM. Des clients peuvent avoir recourt à des services, qui sont en pratique l'exécution d'une fonctionnalité (e.g. multiplication, DCT, encryption), et ce service est réalisé par un, ou plusieurs, serveurs. Les clients et serveurs peuvent être soit des processeurs, soit des coprocesseurs. Pour implémenter ce modèle de programmation, StepNP utilise deux accélérateurs matériels. Premièrement, le MPE (*Message Passing Engine*) permet de convertir les données à envoyer en un format neutre et vice-versa, cette conversion est un requis de l'implémentation puisque DSOC traite avec des éléments hétérogènes. Du côté des processeurs, le MPE est un pont (voir section I.1) reliant le processeur au canal. Grâce au support matériel, la surcharge n'est que de vingt à trente instructions pour un appel DSOC. Le second accélérateur matériel, le ORB (*Object Request Broker*), permet de faire le lien entre les clients et les serveurs. Contrairement au MPE, dont il existe une copie par élément de calcul, le ORB est unique et est un esclave SOCP. L'utilisation de DSOC, en cours de simulation, suit les étapes suivantes :

1. Au début de la simulation, tous les serveurs enregistrent auprès du ORB les services qu'ils desservent.

2. Un client fait appel à un service. Du point de vue de l'application, cette opération se fait exactement de la même manière qu'un simple appel de fonction.

3. Le client envoie à tour de rôle le numéro d'identification de service et les arguments au MPE qui les transfère via le canal au ORB.

4. Par la suite, le client lit la valeur de retour, s'il y en a une, du ORB. Cette requête sera bloquée par le ORB jusqu'à ce qu'il puisse répondre correctement.

Si le service requis n'a pas de valeur de retour, le client poursuit son exécution.

5. Le ORB sélectionne un serveur disponible pour exécuter le service requis et effectue du balancement de charge entre les serveurs. Si aucun serveur n'est disponible, le ORB attend qu'un d'eux le devienne.

6. Le ORB envoie la requête et les arguments au serveur choisi via le canal SOCP et ceux-ci sont récupérés par le MPE.

7. Ensuite, le serveur cherche dans une table de conversion l'adresse de la fonction implémentant le service et saute à cette adresse.

8. Le serveur exécute la fonction implémentant le service.

9. Finalement, le serveur transmet la valeur de retour au ORB via le MPE et le canal SOCP. Le ORB transmet cette valeur de retour au client (qui est toujours en attente) qui la récupère par voie du MPE.

Puisque les opérations réalisées par DSOC sont très similaires à celles du protocole SIDL (*SystemC Interface Definition Langage*) (voir section I.1), les développeurs de StepNP ont réutilisé ce protocole pour implémenter DSOC. Ainsi, les services sont spécifiés en utilisant des classes virtuelles abstraites et le compilateur SIDL génère le code nécessaire aux communications, tant du côté du client que du serveur. Du point de vue de l'utilisateur, DSOC est extrêmement simple, les seules étapes nécessaires sont l'écriture du fichier d'interface, la compilation de l'interface, l'appel de fonction du côté client et l'écriture de la fonction implémentant le service. Bref, les seules étapes desquelles le programmeur doit se soucier sont les étapes deux et huit, toutes les autres lui sont transparentes.

Une limitation de DSOC est que dans StepNP, il y a une correspondance directe entre tous les processus logiques et processus matériels. Ceci implique que lorsqu'un processus se rend disponible pour desservir un service DSOC, il ne peut plus exécuter

une autre partie de l'application. Il peut, par contre, desservir plusieurs services différents.

3.1.1.2 Modèle SMP

Alors que DSOC concerne les systèmes hétérogènes, le deuxième modèle de programmation, SMP avec mémoire partagée, s'adresse aux éléments de calculs homogènes. Comme pour DSOC, ce modèle de programmation est réalisé par la combinaison de deux éléments. Premièrement, un accélérateur matériel, nommé CE (*concurrency engine*), permet de gérer la distribution des processus et l'accès à des ressources partagées, un peu comme le fait le module SISSMA de SPACE. Tout comme le ORB, le CE est unique et est un esclave SOCP. Le deuxième élément est un système d'exploitation léger, nommé Nano-Kernel, qui fournit une interface de programmation (API, *Application Programming Interface*) permettant d'accéder aux fonctionnalités du CE. La combinaison de ces deux éléments permet, par exemple, d'accéder à un sémaphore ou de créer des processus en un simple accès mémoire, indiquant d'un seul coup la ressource concernée et l'opération désirée. Par ailleurs, pour la prise d'un sémaphore, l'accès mémoire sera bloqué par le CE tant que le sémaphore n'est pas libre, ce qui évite d'avoir à gérer un mécanisme de réessaies et de délais.

Comme mentionné précédemment, tous les processus logiques sont associés directement à un processus matériel dans StepNP. Au départ de la simulation, tous les processus matériels disponibles pour l'exécution d'un processus logique, dans un environnement SMP, s'enregistrent auprès du CE et se mettent en attente. Les processus matériels de StepNP sont fixes tout au long de la simulation, mais les processus logiques sont dynamiques. Lorsque l'application désire créer un certain nombre de processus, on écrit une requête au CE, la commande indiquant que l'on veut créer des processus et le nombre de processus sont encodés dans l'adresse exacte

utilisée qui est calculée par le Nano-Kernel. La donnée écrite correspond à l'adresse de la fonction à être exécutée par les processus logiques. Cette dernière opération prend la forme d'un *fork/join* du point de vue de l'application. Par la suite, le CE envoie à tour de rôle l'adresse de la fonction à exécuter aux processus matériels disponibles. Si le nombre de processus à créer est plus grand que le nombre de processus matériels disponibles, le CE attend que ceux-ci se libèrent jusqu'à ce que le nombre de processus demandés ait été exécuté.

Finalement, il est intéressant de noter que l'équipe de StepNP a fait la synthèse du CE et du ORB, démontrant que ceux-ci ne sont pas simplement qu'un concept abstrait, mais qu'ils sont intégrables dans un SoC réel.

3.2 Intégration du Xtensa dans StepNP

Il va de soit que pour simuler l'utilisation du Xtensa dans un MPSoC, il est nécessaire d'inclure à StepNP un modèle de ce processeur. La présente section décrit les travaux antérieurs réalisés sur ce sujet et les ajouts apportés pour la réalisation de ce projet de recherche.

3.2.1 Résumé des travaux antérieurs

Il existe principalement trois approches permettant d'intégrer un ISS à une plateforme SystemC [8]:

1. Réaliser une cosimulation entre SystemC et l'ISS du processeur. Ces deux éléments s'exécutant sur deux processus différents, il est aors nécessaire de créer un pont, connecté au canal de la plateforme, permettant de synchroniser ces deux éléments via des appels distants (RPC, *Remote Procedure Call*).

2. Utiliser un mécanisme de communication déjà existant pour contrôler l'exécution de l'ISS. Dans bien des cas, les ISS sont fournis avec le code source de GDB, qui constitue une interface permettant un tel contrôle.

3. Le code source de l'ISS est directement encapsulé dans un module SystemC.

De ces trois méthodes, la troisième est la plus efficace puisqu'elle permet d'éliminer un niveau d'indirection, facilitant l'intégration de l'ISS et accélérant grandement la vitesse de simulation. Malheureusement, cette méthode n'est pas toujours possible puisqu'elle nécessite la disponibilité du code source de l'ISS.

Tensilica a choisi, pour sa part, de ne pas fournir le code source du processeur Xtensa, confidentialité oblige. En revanche, une bibliothèque en langage C, nommée XTMP (*Xtensa Modeling Protocol*), permet d'accéder aux fonctionnalités de l'ISS. XTMP nécessite également la présence d'un pilote multiprocessus pour fonctionner, mais puisque SystemC en est un, cette restriction ne cause aucun problème.

Par conséquent, étant donné que le code source n'est pas disponible et que Tensilica fournit un API simple vers son ISS, la méthode d'intégration favorisée pour le Xtensa est la deuxième. Un avantage est que la bibliothèque de l'ISS est implémentée en langage C, ce qui facilite l'intégration et les communications. En plus de fournir un accès à l'ISS du Xtensa, XTMP offre la possibilité de modéliser des coprocesseurs et des outils de synchronisation (e.g. sémaphores, mutex), ce qui permet la modélisation d'une plateforme complète.

Un problème important de XTMP, par contre, est que tous les blocs de la simulation doivent être inclus dans un objet de type XTMP_device et doivent suivre le protocole précis de XTMP [38]. Autrement dit, la bibliothèque XTMP est bien adaptée pour créer une plateforme de développement basée sur le Xtensa, mais elle l'est beaucoup moins lorsque vient le temps d'inclure un Xtensa à une plateforme déjà existante.

Une intégration du Xtensa dans StepNP a déjà été réalisée par Bruno Lavigueur lors de ses travaux de maîtrise [38]. Les principaux éléments de cette intégration sont résumés à la figure 3.2. Un pilote SystemC, en faisant appel à l'API de XTMP, permet de créer un nouveaux processeur, lui faire exécuter son programme une instruction par cycle d'horloge et suspendre son exécution. Par ailleurs, l'interface XTMP_device permet de définir les différentes mémoires, ainsi que leur zone d'adresse, qui seront connectées au Xtensa. En plus des mémoires classiques, le Xtensa possède, optionnellement, le port XLMI (*Xtensa Local Memory Interface*) qui permet de connecter des coprocesseurs, modules d'entrée / sortie ou autres éléments adressables de la plateforme qui ne sont pas des mémoires. Étant donnée la nature de ce port, tous les accès mémoires qui y sont faits ne sont jamais mis en cache et on n'y fait aucun accès en rafale. L'interface XTMP_device implémente également des fonctions de rappel pour chacune des zones mémoires présentes. Ces fonctions de rappel sont définies par l'usager et elles seront appellées par l'ISS lorsque celui-ci veut faire un accès mémoire hors de sa cache. Dans l'implémentation présente, les fonctions de rappel définies peuvent soit rediriger ces accès vers une mémoire interne SystemC, vers le canal SOCP ou encore, dans le cas du port XLMI, vers un coprocesseur dédié. L'avantage de la mémoire interne est qu'il est plus facile de modéliser une mémoire générique compatible avec toutes les configurations du Xtensa (i.e taille du bus et ordre des octets). Par contre, cette approche ne permet pas de modéliser les délais associés à un canal de communication, qui sont généralement non négligeables.

3.2.2 Ajouts réalisés

Bien que le module SystemC encapsulant le Xtensa convenait bien aux besoins de la recherche qui fut réalisée, celui-ci a été conçu pour une version antérieure de StepNP ne possédant pas toutes les fonctionnalités décrites précédemment, principalement en ce qui a trait aux caractéristiques MPSoC de StepNP (voir section 3.1.1). Il a

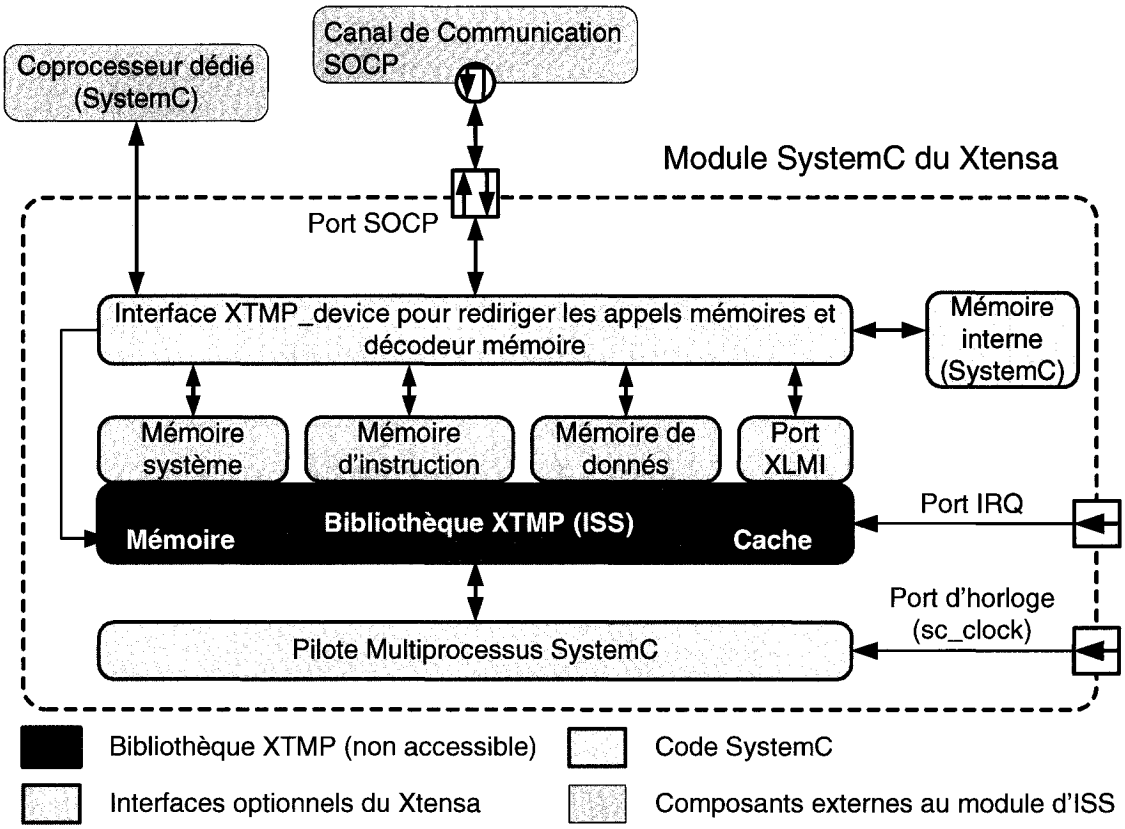

Figure 3.2 Module Xtensa intégré à StepNP

donc été nécessaire d'y apporter certaines modifications pour que le module SystemC corresponde mieux à nos besoins.

Dans le cadre d'une plateforme multiprocesseur, il va de soit que l'on ne peut pas utiliser la mémoire interne du Xtensa comme mémoire centrale du système puisque celle-ci devra être partagée par plusieurs éléments, il est donc nécessaire d'utiliser le canal SOCP pour les accès mémoires. Comme mentionné précédemment, l'utilisation d'une mémoire interne possède l'avantage de pouvoir être générique et permet de s'adapter à toutes les configurations du Xtensa. Ceci n'est pas le cas des divers éléments de StepNP qui sont tous conçus en ayant en tête une architecture 32 bits et petit-boutiste (qui sont les caractéristiques de la machine hôte). Il est tout de même possible d'adapter le module pour supporter toutes les configurations possibles

du Xtensa. Au niveau de l'ordre des octets, dans la fonction de rappel du bloc interface XTMP_device, il est possible d'inverser les octets du mot lorsque le Xtensa est configuré en mode grand-boutiste avant de les transmettre sur le canal SOCP et de les remettre dans le bon ordre au retour. Pour la taille du bus, dans le cas où celle du Xtensa est plus grande que celle de la plateforme, la fonction de rappel peut envoyer un accès en rafale de deux ou quatre mots dépendamment du cas. Bien sur, il y a normalement un plus grand délai associé à un accès en rafale plutôt qu'un accès d'un seul mot, mais ceci constitue une meilleure alternative que de transmettre les mots un par un. De plus, en utilisant le canal avec connexion point à point de StepNP, il est possible d'ajuster la pénalité associée à un accès en rafale, qui peut être mise à zéro. Ceci ne correspond pas à un cas réaliste, mais permet d'obtenir le même résultat qu'en utilisant un bus de 64 ou 128 bits, qui n'est pas disponibles dans StepNP.

Un autre ajout intéressant à faire au module Xtensa est de supporter les ponts de StepNP prévus pour les processeurs ARM et PowerPC. L'utilisation de ces ponts fourni un API donnant accès à plusieurs fonctionnalités utiles comme l'affichage de messages à l'écran ou la pose de points d'interruption. De plus, le support des ponts est nécessaire pour supporter le MPE et utiliser les outils de passage de messages de StepNP. Un autre point est que les ponts, par héritage, rendent le Xtensa compatible avec l'outil de génération dynamique SocGen (*SoC Generation*), permettant ainsi de créer une plateforme contenant des Xtensa à partir d'un script Python. Par ailleurs, les fonctionnalités de SocGen permettent de communiquer avec le Xtensa en cours de simulation, via SIDL, de manière à obtenir des informations utiles, comme par exemple le contenu des registres, ou pour envoyer des commandes, comme par exemple de charger un programme ou de connecter un dévermineur.

3.2.2.1 Processus d'écriture et de lecture

Les délais des composants SystemC (e.g. canal de communication, coprocesseurs) dans la plateforme StepNP ont posé un problème lors de l'intégration du Xtensa. La bibliothèque XTMP fournit une fonction, `XTMP_wait`, permettant d'interrompre momentanément l'exécution d'un processeur. Cette fonction, entre autres choses, appelle la fonction `sc_wait` de SystemC qui cause un délai dans le processus appelant. Le problème est que `XTMP_wait`, au moment de son appel, doit connaître le nombre exact de cycles de délai. Cette fonction ne permet pas non plus d'attendre sur un signal, comme on peut le faire avec `sc_wait`. Or, StepNP fait abondamments usage de l'attente sur un signal, principalement pour synchroniser les processus dans les environnement SMP et DSOC et pour suspendre l'exécution des processeurs ou coprocesseurs lors d'un accès mémoire. Un autre problème rencontré est qu'il est impossible de suspendre l'exécution du Xtensa lors d'une écriture, celui-ci assume que la requête sera mise dans un tampon et qu'elle sera gérée pendant que le processeur poursuit son exécution. Par contre, le protocole SOCP de StepNP bloque l'appel lors d'accès mémoire, tant en lecture qu'en écriture, tans que l'accès mémoire n'est pas complété. Il est également impossible de simplement utiliser la fonction `sc_wait` plutôt que `XTMP_wait` car le temps vu par le processeur Xtensa et le temps vu par SystemC deviennent désynchronisés. Ces contraintes ne présentent pas de problèmes majeurs pour quelqu'un désirant construire une plateforme basée sur XTMP, mais malheureusement ces contraintes n'étaient pas présentes au moment de développer StepNP. Voilà donc un exemple illustrant qu'il n'est pas nécessairement aisé d'adapter la bibliothèque XTMP à une plateforme déjà existante et où les détails d'implémentation interne de cette bibliothèque seraient bien utiles.

Pour contourner cet obstacle, nous avons ajouté deux processus SystemC au module Xtensa, en plus du processus déjà existant, pour contrôler l'exécution de l'ISS. Le principe de base de ces trois processus est illustré à la figure 3.3 qui présente le

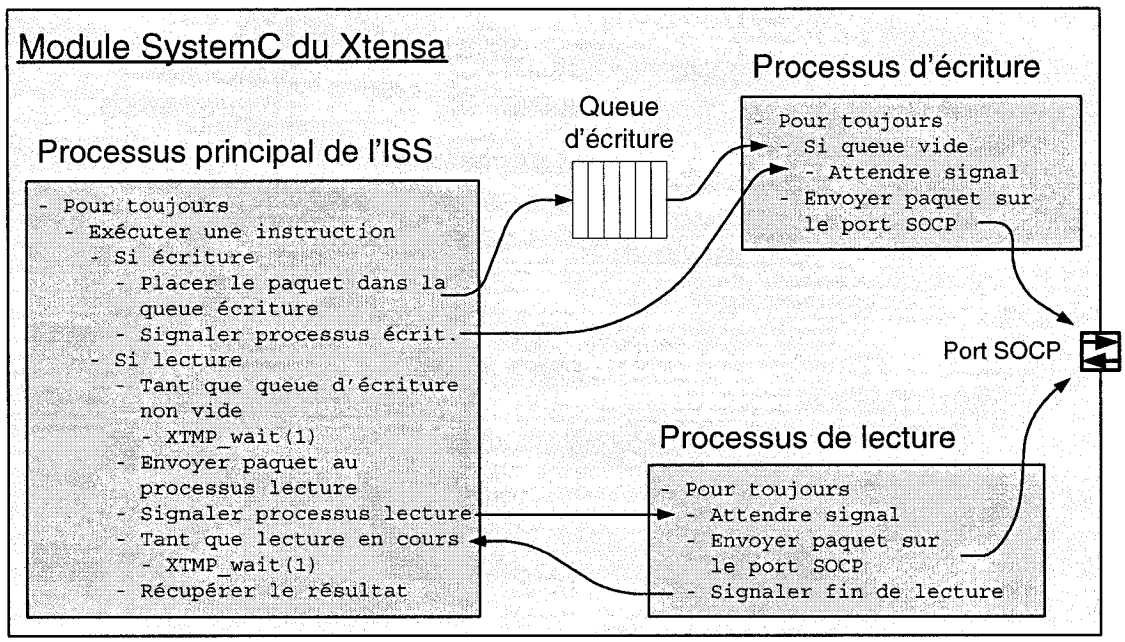

Figure 3.3 Les trois processus SystemC du module Xtensa

pseudocode de ces processus et la synchronisation qui les accompagne.

Comme c'était déjà le cas, un processus SystemC permet de faire avancer le programme du Xtensa cycle par cycle. Des modifications ont été apportées aux fonctions de rappel lors d'une écriture ou d'une lecture. Comme il a été mentionné précédemment, il est interdit de suspendre l'exécution de l'ISS lors d'une écriture. Pour répondre à ce besoin, un deuxième processus s'occupe des écritures du Xtensa. Lors d'une écriture, le processus de l'ISS place tout simplement le paquet SOCP dans une queue de messages, de profondeur dix, signale le processus d'écriture qu'un nouveau paquet est arrivé et poursuit son exécution, évitant ainsi toute attente. De son côté, le processus d'écriture lit les paquets de la queue et les envoie un par un sur le canal SOCP. Puisque le processus d'écriture n'est pas soumis aux mêmes règles que le processus de l'ISS, les délais du canal SOCP ne posent aucun problème et il est également possible d'utiliser les constructions de SystemC pour synchroniser ces deux processus.

L'opération de lecture est légèrement plus complexe puisque le processeur ne peut pas simplement transférer la requête et poursuivre. Comme dans le cas de l'écriture, un processus est ajouté pour gérer les lectures. Le processus contrôlant l'ISS et celui de lecture sont synchronisés suivant un mécanisme de rendez-vous bilatéral. C'est à dire que ces deux processus s'attendent mutuellement l'un l'autre. Comme pour le cas du processus d'écriture, le processus de lecture peut attendre sur un signal SystemC. Le processus de l'ISS de son côté utilise une stratégie d'attente active (*pooling*), c'est à dire que celui-ci vérifie l'état d'une variable de synchronisation à chaque cycle. Ceci nous permet d'utiliser la fonction `XTMP_wait` pour suspendre l'exécution de l'ISS un cycle à la fois, ce qui garde l'ISS parfaitement synchronisé avec la plateforme SystemC. Pour préserver l'ordre des accès mémoire, c'est à dire d'éviter qu'une lecture ait lieu avant les écritures contenues dans la queue, il est nécessaire, lors de l'opération de lecture, d'attendre que toutes les écritures en suspend soient terminées, autrement dit que la queue soit vide. Ceci est accompli en utilisant encore une fois une stratégie d'attente active. Cette attente cycle par cycle de l'ISS présente la principale faiblesse de l'implémentation réalisée puisqu'elle engendre une grande quantité de changements de contextes, ce qui a pour effet de dégrader les vitesses de simulation. Par contre, l'expérience nous a indiqué que l'augmentation du temps de simulation reste inférieure au double.

3.2.2.2 Gestion de la mémoire et de la cache

La gestion de la mémoire et de la cache présente également des difficultés, à deux niveaux, pour l'intégration du Xtensa dans StepNP. En premier lieu, puisque dans un environnement SMP la mémoire est partagée entre plusieurs processeurs, la porte est ouverte aux problèmes de cohérence de cache. En deuxième lieu, les accès mémoire de StepNP dirigés vers les coprocesseurs, comme le CE et le ORB, ne doivent jamais être mis en cache, premièrement parce qu'ils doivent toujours se rendre au coprocesseur et

parce qu'une cache implique des requêtes en rafale lors du chargement d'une nouvelle ligne. Pour régler ce premier problème, nous avons d'abord considéré créer un Xtensa sans cache et gérer celle-ci à l'aide des caches déjà incluses dans StepNP. Par contre, cette approche est impossible en pratique car même si le Xtensa n'a pas de cache, il nécessite tout de même plusieurs cycles de pénalité pour chaque accès mémoire. L'expérience montre que lorsque le Xtensa est généré sans cache, environ quarante cycles par accès mémoire sont perdus avant même que l'accès sorte de l'ISS, c'est à dire en excluant les délais relatifs aux autres éléments de la plateforme dont le canal de communication. N'ayant pas accès aux détails d'implémentation de l'ISS, il est difficile de déterminer avec certitude l'origine de ces cycles, mais une hypothèse serait que l'ISS, même en l'absence d'une cache, simule tout de même un contrôleur de cache utilisant plusieurs cycles par accès mémoire. Il est donc nécessaire d'utiliser la cache incluse avec l'ISS pour obtenir des résultats de simulation acceptables.

Au sujet du deuxième problème, les coprocesseurs, ceci devrait se résoudre en passant par le port XLMI du Xtensa prévu justement à cette fin. L'inconvénient est que le port XLMI supporte au maximum une plage d'adresses de 256 kilo-octets. Or, les éléments de StepNP nécessitent une bien plus grande plage d'adresses, le ORB à lui seul nécessite seize mégaoctets. La raison pour laquelle les éléments de StepNP utilisent une si grande plage d'adresses est qu'une grande quantité d'information est encodée dans l'adresse (e.g. commande à effectuer, identificateur du service demandé, identificateur des éléments concernés), ce qui contribue à l'obtention d'une plage d'adresses gigantesque pour un coprocesseur.

L'espace mémoire total du Xtensa, codé sur 32 bits d'adresse, est partagé en huit zones de tailles égales. Il est possible, pour chacune de ces zones, d'ajuster le mode d'accès à la cache. Afin de pouvoir profiter de la cache du Xtensa et des coprocesseurs de StepNP, nous avons donc désactivé la cache du Xtensa pour certaines zones mémoire causant problème. La figure 3.4 et le tableau 3.1 résument l'état des différentes zones

Tableau 3.1 Zones mémoire du Xtensa

Mémoire	Départ	Fin	Contenu
SRAM	0x60000000	0x68000000	Stack, .text
SROM	0x40000000	0x41000000	Non utilisée
Monceau	0x28040000	0x40000000	Monceau (*heap*)
DRAM	0x28000000	0x28040000	.bss, .data
StepNP	0x10000000	0x22000000	ORB, CE, MPE et autres

mémoire.

Premièrement, la plage d'adresses occupée par les outils de StepNP nous force à désactiver la cache pour les deux premières zones mémoire. Il est à noter que StepNP n'utilise qu'un petite partie de cette plage, mais puisque les adresses utilisées chevauchent les deux zones, le résultat est le même. Cette plage d'adresses ne correspond à aucune mémoire du point de vue du Xtensa, celui-ci croit qu'elle est vide. Par contre l'ISS ne semble pas avoir de problème à ce que le programme écrive à une adresse non attribuée, à condition d'y avoir connecté un module. Ceci nous amène donc à placer les mémoires devant avoir une cache associée à des adresses supérieures à 0x40000000. La mémoire ROM principale peut évidemment être mise en cache, étant donné qu'aucune écriture ne devrait y être faite. Ceci ne change rien par contre puisque le Xtensa n'utilise pas la ROM à moins qu'il en soit spécifié autrement.

Par défaut, la mémoire RAM principale, qui est dans une zone où la cache est activée, contient la pile, le monceau (*heap*) et les segments de code .text, .data et .bss. L'implémentation SMP de StepNP est telle que tous ces segments sont partagés par tous les processeurs à l'exception de la pile qui est propre à chaque processeur. Il a donc été décidé de laisser la pile dans une zone où la cache est activée, plus particulièrement la mémoire RAM principale, puisque celle-ci ne présente pas de problème de cohérence. Par contre, il est nécessaire de s'assurer qu'un processeur ne passe pas l'adresse d'une variable allouée sur sa pile à un autre processeur, cette erreur

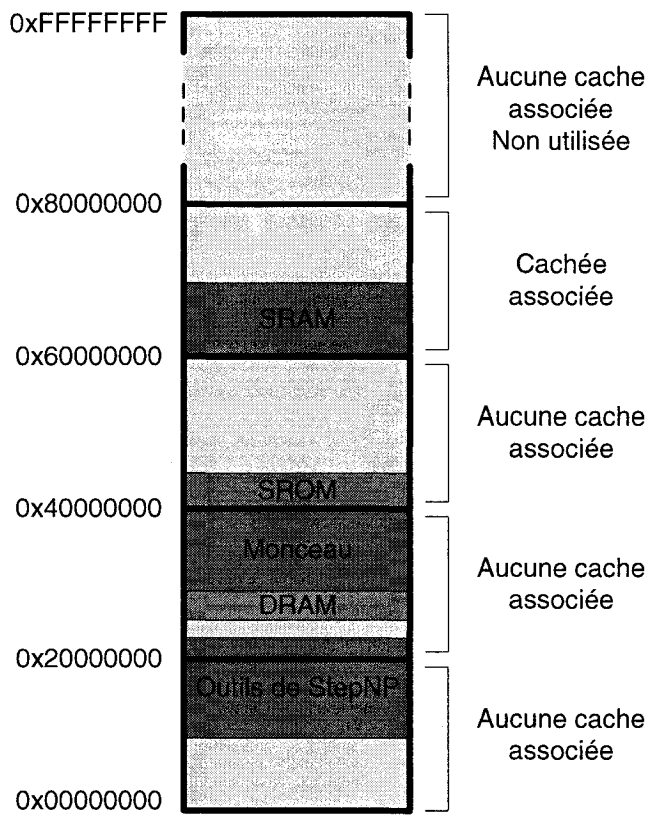

Figure 3.4 Zones mémoire du Xtensa

est facile à faire avec une implémentation DSOC ou SMP où lancer un processus est très similaire à un appel de fonction.

Le monceau est contrôlé par le nano-kernel de StepNP, qui redéfinit son emplacement et les fonctions d'allocation (`malloc`) et de libération (`free`) de mémoire. Il conserve également des variables globales, protégées par le CE, permettant d'indiquer les espaces mémoire libres. Comme les variables allouées sur le monceau sont généralement globales et partagées, celui-ci a été placé dans une zone mémoire où la cache est désactivée. Puisque l'application, par voie du Nano-Kernel, contrôle l'emplacement du monceau, il a été placé dans une zone ne correspondant à aucune mémoire du point de vue du Xtensa, comme dans le cas des outils de StepNP. Cette mesure permet de s'assurer de ne pas écraser accidentellement une zone utilisée par

le compilateur.

Les segments .bss et .data, comme pour le cas du monceau, contiennent des variables partagées par plusieurs processeurs et seront mis dans une zone où la cache est désactivée. Dans leur cas par contre, l'application ne contrôle pas leur emplacement. Nous avons donc ajouté une mémoire au Xtensa, une RAM de données, dans une des zones mémoire où la cache est désactivée. Au moment de la compilation du programme, un script de liaison nous permet de placer les segments .bss et .data dans la mémoire RAM de données. Finalement, le segment .text contient les instructions du programme à exécuter. Puisque ce segment ne devrait jamais être modifié, il est demeuré en mémoire RAM principale et donc dans une zone mise en cache.

L'implémentation utilisée, bien que fonctionnelle, présente un certain problème : une grande partie de la mémoire n'est jamais mise en cache. Ceci présente le risque de dégrader considérablement les résultats de performance. Pour donner une idée, le graphique de la figure 3.5 présente la proportion des accès mémoire de chaque zone identifiées précédemment pour l'application d'encodage MPEG-4 qui sera étudiée au prochain chapitre. Les proportions présentées varient en fonction de certains paramètres de simulation, mais elles permettent tout de même d'avoir une approximation du coût de cette approche. On constate donc que pour cette application, 87% (pile + code) des accès mémoire seront dans une zone mise en cache. Les accès aux composants spéciaux de StepNP (e.g. CE, ORB) présentent une proportion négligeable et la zone identifiée « réservée » sera discutée lors des détails de la plateforme d'encodage.

Un dernier élément ayant fait obstacle à l'intégration du Xtensa dans la nouvelle version de StepNP fut la différence de version de compilateur entre StepNP, compatible avec GCC version 3.2, et la bibliothèque XTMP, compatible avec GCC 2.95. Devant l'impossibilité d'obtenir une version plus à jour des outils de Tensilica,

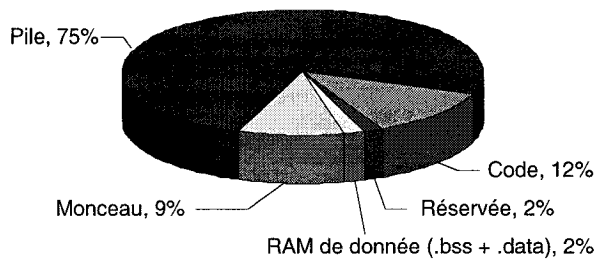

Figure 3.5 Distribution des accès mémoire

il a fallu se tourner vers la modification du code source de StepNP pour le rendre compatible avec la version 2.95 de GCC. Les détails de cette tâche ne seront pas discutés ici puisqu'ils ne présentent pas beaucoup d'intérêts.

3.2.3 Améliorations possibles

Plusieurs améliorations seraient possibles au niveau du module SystemC du Xtensa. En premier lieu, comme il a été mentionné, les attentes constantes de un cycle à la fois, reliées au processus d'écriture et de lecture, causent un ralentissement de la vitesse de simulation. Une solution serait d'augmenter la durée des suspensions lorsque celles-ci deviennent trop longues. Un simple accès mémoire se résout typiquement en une douzaine de cycles environ. Donc si l'attente se prolonge au delà de ce nombre, on a fort probablement affaire à un serveur DSOC ou un processus SMP qui est en attente indéfinie, celle-ci pouvant prendre des milliers de cycles. Dans ce cas, il pourrait être avantageux d'attendre par coups de dix cycles (valeur arbitraire) plutôt que un. Un mécanisme d'interruption constitue également une autre avenue qui vaut la peine d'être explorée.

Comme il est expliqué à l'annexe I, StepNP fourni plusieurs outils facilitant l'introspection des processeurs. Ces outils, basés sur SIDL, permettent d'inclure une sonde à chaque processeur. Bien que l'ISS du Xtensa ne permette pas d'obtenir tous les détails internes du processeur, certains éléments d'introspection pourraient être

attribués à des sondes et ainsi récupérés par SocMon (*SoC Monitor*) ou un script Python. Comme par exemple la charge moyenne de travail, qui pourrait être obtenue du le temps total d'exécution et le temps passé par le processeur à attendre sur des accès mémoire.

Finalement, un meilleur mécanisme de cache serait probablement de mise. Pour les zones mémoire où la cache du Xtensa est activée, il n'y a aucun problème. Par contre, pour celles où elle est désactivée (monceau et segments .bss et .data), le processeur perd beaucoup de cycles de lecture pour rien. Le modèle du Xtensa utilisé ne semble pas permettre d'explicitement invalider une ligne de cache à partir de la bibliothèque XTMP, mais il serait tout de même possible de modéliser une cache en SystemC. Bien que cette solution ne soit pas idéale, puisqu'une cache modélisée en SystemC sera toujours plus lente que celle incluse avec l'ISS, ceci constitue tout de même une meilleure approche que de ne rien faire.

CHAPITRE 4

CONCEPTION D'UNE PLATEFORME D'ENCODAGE MPEG-4 POUR STEPNP

Dans ce dernier chapitre, nous nous intéressons à la modélisation d'une plateforme d'encodage MPEG-4 utilisant StepNP. Nous décrirons dans un premier temps la plateforme réalisée. Dans un deuxième temps, nous présenterons les résultats de simulation obtenus, plus particulièrement ceux concernant l'utilisation de plusieurs processeurs configurables.

4.1 Description de la plateforme

Comme il a été expliqué au chapitre précédent, StepNP est divisé en trois environnements de travail (i.e matériel, logiciel et outils). Pour suivre cette caractéristique, la plateforme d'encodage sera décrite en deux parties, soit la plateforme matérielle, contenant tous les composants présents, et la plateforme logicielle, illustrant la parallélisation de l'application sur les différentes unités de traitement et le flot de l'application. Les outils SoC ne seront pas décrits puisque aucun ajout important n'a été fait aux outils existants.

4.1.1 Plateforme matérielle

Les principaux éléments de la plateforme matérielle sont illustrés à la figure 4.1. En premier lieu, on retrouve un groupe de processeurs effectuant le traitement de l'application. Il est possible d'utiliser soit des processeurs ARM, déjà inclus à la

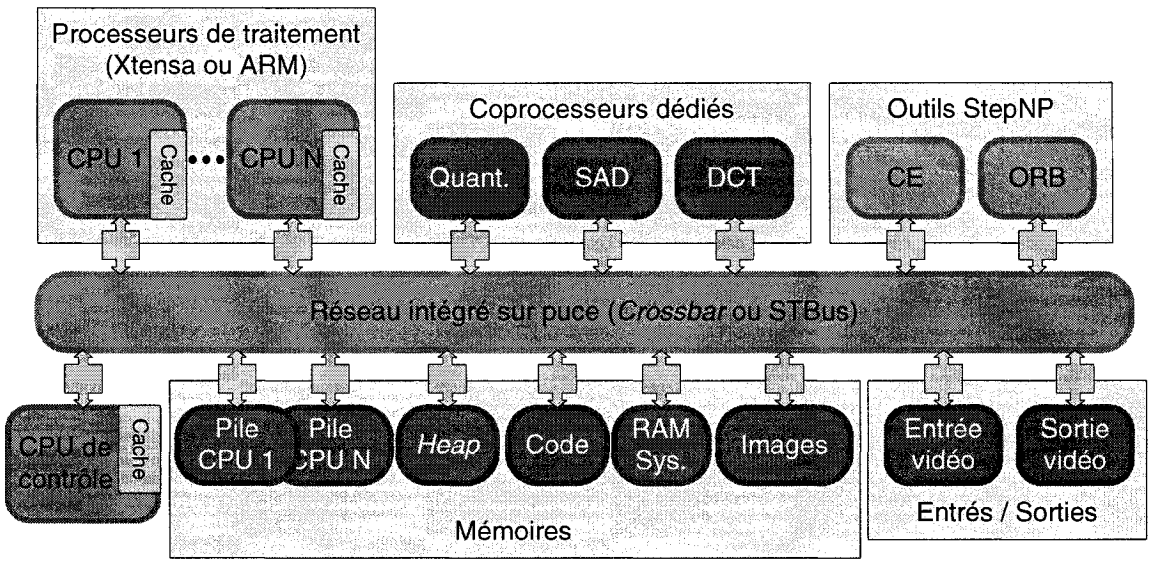

Figure 4.1 Plateforme d'encodage MPEG-4

plateforme StepNP, ou encore des processeurs Xtensa, dont l'intégration a été décrite au chapitre précédent. Une contrainte présente, par contre, est que pour utiliser le modèle SMP, considérant que le code et le monceau sont partagés, il est nécessaire que tous les processeurs exécutent exactement le même programme. Ceci implique donc qu'il est impossible de mélanger des processeurs ARM et Xtensa et que si nous utilisons des Xtensa, ceux-ci doivent avoir les mêmes instructions spécialisées. Un processeur supplémentaire est également nécessaire pour synchroniser les différents processus de l'application. Puisque nous utilisons le modèle SMP, ce processeur doit exécuter le même code que les autres et il doit donc être identique aux autres. Par contre, il effectue une quantitée de calculs négligeable par rapport aux processeurs de traitement.

La plateforme possède également trois coprocesseurs dédiés, qui sont en pratique des serveurs DSOC, pouvant être utilisés pour accélérer une partie de l'application. Les services disponibles sur ces coprocesseurs sont les suivants :

DCT : DCT, DCT inverse, zigzag, zigzag inverse, addition et différence de blocs

SAD : SAD

Quant. : Quantification et quantification inverse

Les coprocesseurs de la plateforme ont été modélisés avec certains atouts les rendant très efficaces. Premièrement ils disposent d'une mémoire locale permettant d'emmagasiner une image, réduisant le taux d'accès mémoire. Deuxièmement, ils sont en mesure d'effectuer des accès en rafale d'une très grande taille sur le canal de communication.

Pour produire une simulation plus réaliste, deux modules ont été ajoutés pour modéliser les entrées et sorties de la plateforme. Plus concrètement, l'entrée vidéo lit le fichier à encoder directement de la mémoire de la machine hôte et le place dans la mémoire de la plateforme. À l'inverse, la sortie vidéo lit le fichier encodé de la mémoire de la plateforme et l'écrit dans celle de la machine hôte.

Le système est également accompagné de plusieurs mémoires correspondant principalement aux différentes mémoires identifiées à la section 3.2.2.2. Pour limiter la congestion reliée à un trop grand nombre d'accès à une même destination, chaque processeur possède une mémoire dédiée pour sa pile. La mémoire d'images est une zone utilisée par le bloc d'entrée vidéo pour écrire les images lues à partir de la machine hôte. Cette dernière zone correspond également à la zone " réservée " dont il avait été question à la fin de la section 3.2.2.2.

Finalement, tous ces éléments sont reliés ensemble par un NoC de type connexion point à point (*crossbar*). Des améliorations possibles incluent l'utilisation de topologies ayant une latence faible entre deux éléments rapprochés, comme par exemple le RoC [23], pour tirer profit du haut débit entre un processeur et sa pile.

4.1.2 Plateforme logicielle

Le flot suivi par l'application d'encodage MPEG-4 est décrit à la figure 4.2. Le tout démarre avec le bloc d'entrée vidéo, qui est synchronisé pour lire une image à partir du fichier original à chaque trentième de seconde. Une fois que l'image est écrite en mémoire, ce bloc envoie une requête DSOC au processeur de contrôle, qui est le serveur, pour qu'il commence son traitement. Dans le cas où l'encodage de l'image précédente ne serait pas complété, l'appel DSOC ne sera pas traité immédiatement. Par contre, cette situation ne cause aucun problème puisque l'appel sera tout simplement suspendu jusqu'à ce que le processeur puisse répondre. On peut noter également qu'une implémentation où le processeur de contrôle est le client et l'entrée vidéo le serveur aurait été tout à fait valable et aurait donné les mêmes résultats.

Une fois l'encodage d'une image démarré par le processeur de contrôle, l'application est séparée en plusieurs processus s'exécutant de manière concurrente. La parallélisation de l'application est effectuée au niveau des macroblocs. Pour chacune des opérations (estimation de mouvement, DCT et les autres), le processeur de contrôle lance une série de processus, grâce au modèle de programmation SMP de StepNP, qui sont exécutés par les processeurs de traitement et qui effectuent chacun l'opération courante d'encodage sur un macrobloc différent. Le processeur de contrôle s'assure également que l'ordre dans lequel les processus sont lancés respecte toutes les dépendances de données entre les macroblocs. Dans la version utilisée, le processeur de contrôle attend que tous les processus lancés aient terminé leur exécution avant de poursuivre, par contre il serait possible de modifier l'application pour faire participer ce processeur au traitement.

Trois versions de l'application ont été modélisées. En premier lieu, la version entièrement logicielle utilise des processeurs RISC de base pour réaliser la totalité

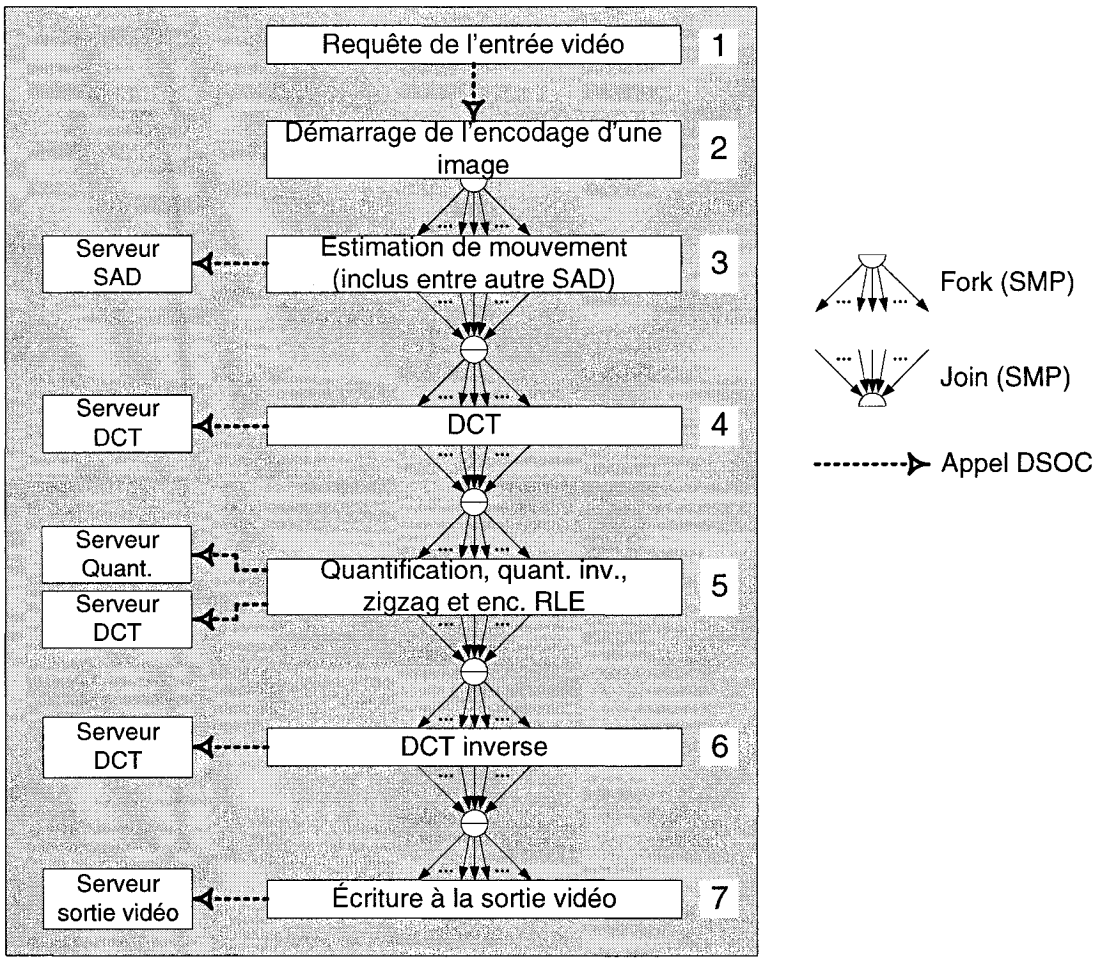

Figure 4.2 Flot de contrôle de l'application MPEG-4

des calculs. La seconde version, quant à elle, utilise des processeurs RISC combinés à des coprocesseurs matériels pour accélérer les points critiques de l'application. Les coprocesseurs sont les blocs DCT, SAD et Quant, tous de type serveur DSOC. Finalement, la dernière version utilise des processeurs Xtensa auxquels nous avons ajouté les instructions spécialisés détaillées à la section 2.4.1. Il est difficile d'effectuer une bonne comparaison entre le processeur ARM et le processeur Xtensa dans StepNP car ces deux processeurs ne sont pas modélisés avec le même niveau de détail, particulièrement au niveau des délais associés aux accès mémoires et à la cache. Pour contourner ce problème, les simulations impliquant un processeur RISC de base ont été faites en utilisant un Xtensa sans aucune instruction spécialisée, ce qui correspond

en soit à un processeur RISC de base.

4.1.3 Contributions

Afin de bien clarifier les contributions de ce travail au niveau du développement de la plateforme MPEG4, nos principals réalisations sont les suivantes:

- la modélisation de l'entrée vidéo,
- l'intégration du modèle du StBus à la plateforme,
- le développement du coprocesseur de quantification,
- l'entrelacement de la zone d'adresse sur une banque de mémoires (non discuté ici).

Nous devons à STMicroelectronics le développement du reste de la plateforme, dont la tâche la plus importante est la distribution de l'application sur plusieurs processus tout en respectant les dépendances de données.

4.2 Résultats de simulation

La présente section compare les résultats de simulation des trois versions de l'application pour trois cas de tests différents. Puisque les simulations sont relativement longues (plusieurs heures par images) nous nous sommes contentés de ne simuler que l'encodage de deux images. Les résultats de simulation présentés sont pour la deuxième image du fichier, puisque la première suit un encodage différent des 29 autres et ne présente pas une idée juste de l'encodage global. D'un autre côté, les délais d'encodage des images 2 à 30 sont comparables, ce qui nous donne une bonne représentation du temps d'encodage total du vidéo. Finalement, certains problèmes d'optimisation lors de la compilation nous empêchent de faire une bonne comparaison

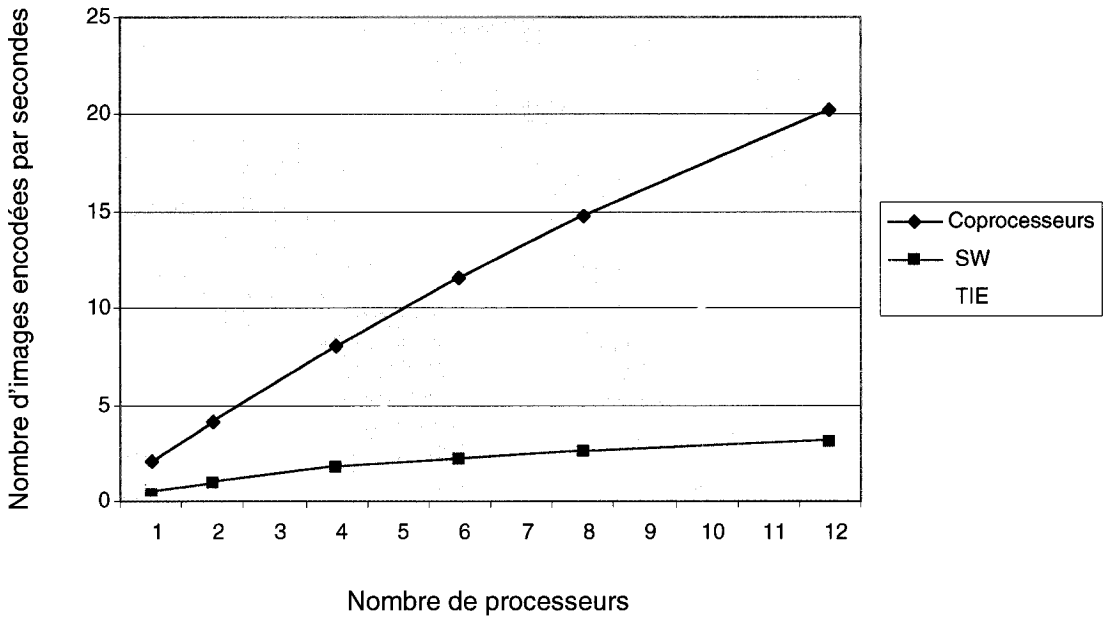

Figure 4.3 Vitesse d'encodage pour une latence de communication nulle

avec les résultats de la section 2.4.2. Lorsque l'application d'encodage est compilée en n'utilisant aucune optimisation, tout fonctionne correctement. Par contre, lorque la compilation est faite avec optimisation, l'application se termine brusquement suite à une faute d'assertion. Nous n'avons malheureusement pas pu résoudre ce problème, mais nous croyons que le compilateur a réordonnancé les instructions de manière à ce qu'un processus soit lancé avant qu'une variable qu'il utilise n'ait été initialisée.

4.2.1 Latence de communication nulle

Ce premier test a été réalisé en utilisant un canal de type connexion point à point avec une latence de communication nulle. Les résultats de simulation sont présentés à la figure 4.3. Comme on pouvait s'y attendre, les versions avec instructions TIE et avec coprocesseurs sont plus rapides que la version entièrement logicielle. Un peu plus décevant par contre, la version avec coprocesseurs est plus rapide que la version avec instructions spécialisées. Ceci peut s'expliquer par l'efficacité notable des mémoires

Tableau 4.1 Identification des maîtres et esclaves

Esclaves		Maîtres	
ID	**Description**	**ID**	**Description**
0	Mémoire RAM	0	Processeur de contrôle
1	Mém. ROM (non utilisée)	1 .. 4	Processeurs de traitement
2	Mémoire d'images	5	Serveur DNS (pour DSOC)
3	Mémoire de données	6	Coprocesseur DCT
4	Monceau	7	Coprocesseur SAD
5	CE	8	Coprocesseur Quant.
6	ORB	9	Sortie vidéo
7	Pile du processeur de contrôle	10	Entrée vidéo
8 .. 11	Pile des proc. de traitement	11+	Non utilisés

locales des coprocesseurs pour limiter les accès sur le canal. La figure 4.4 donne le nombre de transactions effectuées lors d'une simulation à quatre processeurs. Pour suivre le tout, le tableau 4.1 donne la correspondance entre les numéros d'identification des maîtres et esclaves et leur description. On observe, à partir du graphique des accès mémoires de la figure 4.4, que la version TIE de l'application effectue environ le tiers du nombre d'accès de la version originale sans accélérateurs (environ 13 millions par processeurs versus 35 millions par processeurs). Il va de soi également que dans la version avec coprocessseurs, les processeurs effectue encore moins d'accès puisque la grande partie du traitement est effectuée par d'autres éléments de calcul (environ 7 millions par processeurs). On observe par contre que la version avec instruction TIE effectue seulement un tier du nombre d'accès mémoire que la version

Un autre élément intéressant à observer est le facteur d'accélération des trois versions de l'application en fonction du nombre de processeurs présents. Le graphique de la figure 4.5 donne l'état des accélérations, pour les trois versions, en fonction du résultat pour un processeur avec la version correspondante. On observe dans un premier temps que pour la version logicielle, cette accélération ne croît pas très bien avec l'augmentation du nombre de processeurs. La fonction limitant l'accélération pour cette version est celle d'estimation de mouvement. Ceci peut s'expliquer par le

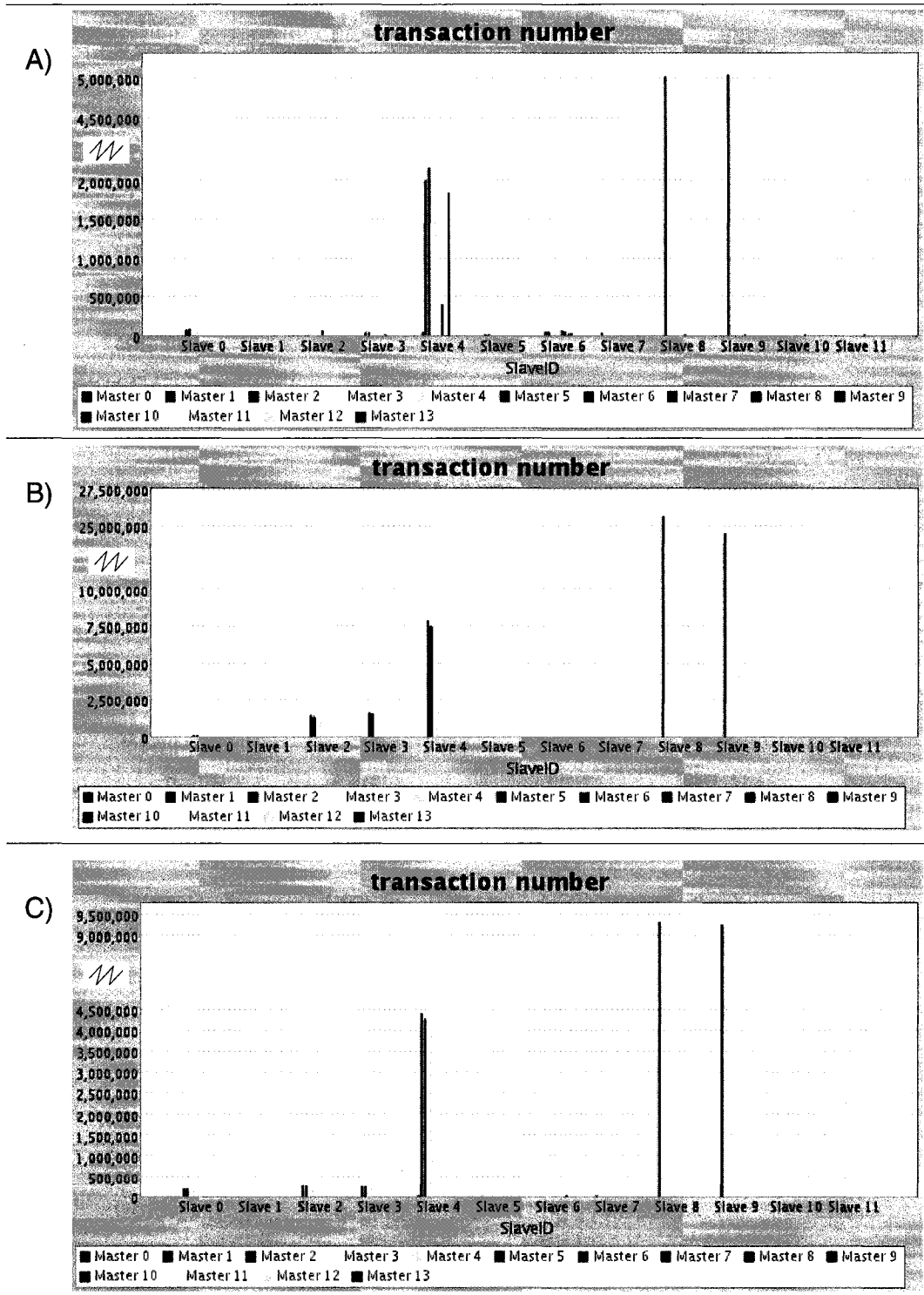

Figure 4.4 Nombre de transactions pour une simulation à quatre processeurs. A) Cas des coprocesseurs. B) Cas entièrement logiciel. C) Cas des instructions spécialisées

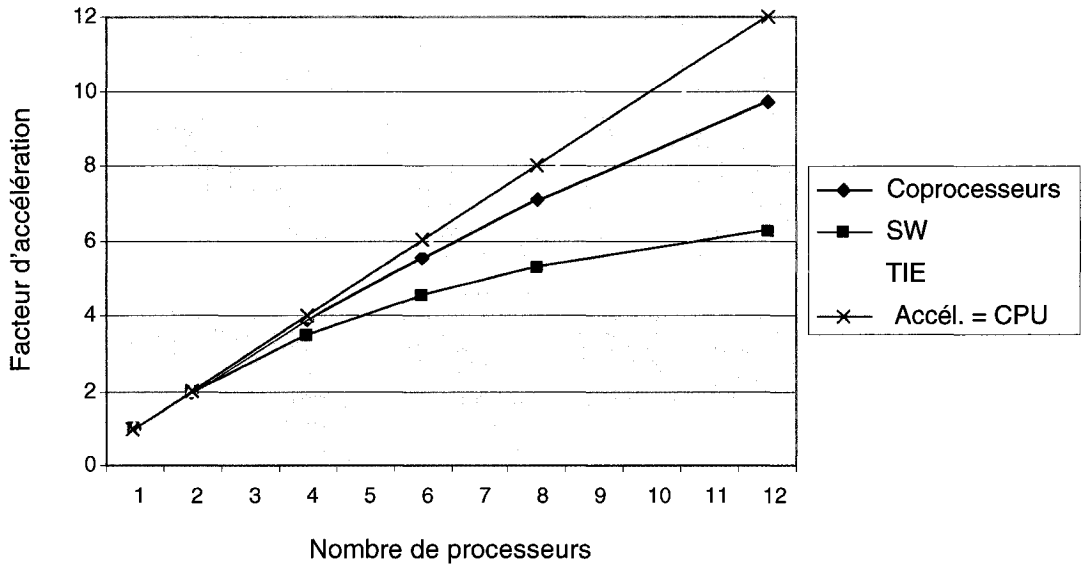

Figure 4.5 Facteur d'accélération en fonction du nombre de processeurs

fait que les processus sont lancés par groupes plus petits que les autres fonctions afin de respecter les dépendances. Ceci fait en sorte que plusieurs processeurs se doivent d'attendre la fin de tous les processus avant de pouvoir continuer, réduisant leur taux d'utilisation. Puisque les versions avec TIE et avec coprocesseurs sont plus rapides, l'attente est moins longue et ce problème est moins apparent.

Plus intéressant par contre, l'accélération de la version avec TIE est meilleure que celle de la version avec coprocesseurs. Ceci vient de l'augmentation du taux d'utilisation des coprocesseurs qui deviennent le goulot d'étranglement de l'application lorsqu'ils sont fortement sollicités. La version avec instruction TIE ne partage pas le même problème, ce qui lui permet d'atteindre une meilleure accélération avec l'augmentation du nombre de processeurs.

Tableau 4.2 Délai d'encodage d'une image avec et sans latence

	4 CPU			12 CPU		
	Sans lat.	Avec lat.	Décélération	Sans lat.	Avec lat.	Décélération
Copro.	8.08	5.54	0.69	20.23	11.37	0.56
SW	1.71	1.13	0.66	3.06	1.98	0.65
TIE	4.20	3.03	0.72	11.94	8.65	0.72

4.2.2 Canal avec latence sans contention

Des tests ont également été réalisés sur un canal ayant une latence de communication de trois cycles pour un accès complet, ce qui correspond à la latence minimale du StBus lorsqu'il n'y a pas de contention. Le tableau 4.2 présente le délai, en microsecondes, pour l'encodage d'une image ainsi que le facteur de diminution de performance relié à l'ajout de latence sur le canal de communication.

Bien évidemment, la vitesse de l'application est réduite dans tous les cas, suite à l'ajout d'une latence. Il est intéressant d'observer que la version qui semble la moins sensible à cette augmentation de la latence est celle avec instructions TIE. Par rapport à la version logicielle, la version avec instructions TIE réduit considérablement le nombre d'accès mémoires, ce qui explique que cette version soit moins sensible à la latence. Du côté de la version avec coprocesseurs, puisque tous les appels vers ceux-ci sont fait par des accès mémoires, qui sont en double puisque l'on doit passer par le ORB, il est normal que le temps de réponse soit fortement influencé par la latence du canal. On ne s'étonne donc pas de voir que l'utilisation d'instructions spécialisées rendent notre plateforme moins sensible à la latence de communication.

4.2.3 Canal avec latence et contention

Finalement, un troisième cas de test a été réalisé pour simuler l'effet d'un canal avec contention. Pour ce test, nous avons utilisé le même canal de communication point

Tableau 4.3 Résultats de simulation avec contention sur le canal

	1 CPU	3 CPU	Accel.	10 CPU	Accel.
Copro.	1.19	2.83	2.38	4.60	3.87
SW	0.26	0.61	2.31	0.92	3.53
TIE	0.63	1.57	2.49	2.95	4.69

à point, mais il a été modifié pour avoir une latence de quatre cycles plus 20% par transaction présente sur le canal au moment de lancer la requête. Les résultats de ce troisième cas de test sont présentés au tableau 4.3.

Le premier élément qui ressort de ce tableau est que l'accélération ne croit pas très bien avec le nombre de processeurs pour les trois versions de l'application. Cette situation est particulièrement évidente pour le cas de test à dix processeurs. Ceci s'explique facilement par la grande pénalité associée à la présence de plusieurs paquets sur le canal. Les résultats présentés ici confirment que l'utilisation d'instructions spécialisées procure une implémentation moins sensible aux délais du canal de communication que l'utilisation de coprocesseurs. Cette situation est mise encore plus en évidence avec la simulation de la contention sur le canal. Dans le cas de la version avec coprocesseurs, étant donné qu'il y a un plus grand nombre d'éléments émettant des requêtes simultanément sur le canal, il est normal que la contention ait un effet plus grand. Le nombre plus élevé d'éléments connectés au canal entraînerait également une plus grande distance entre ceux-ci dans certaines architectures de NoC, comme par exemple le RoC. Ceci contribuerait encore une fois à réduire les performances de la version avec coprocesseurs par rapport à celle avec instructions spécialisées.

4.3 Améliorations possibles

D'autres types de distribution mériteraient d'être évaluées pour l'encodage MPEG-4. Dans notre cas, chaque opération doit être complétée pour tous les macroblocs avant

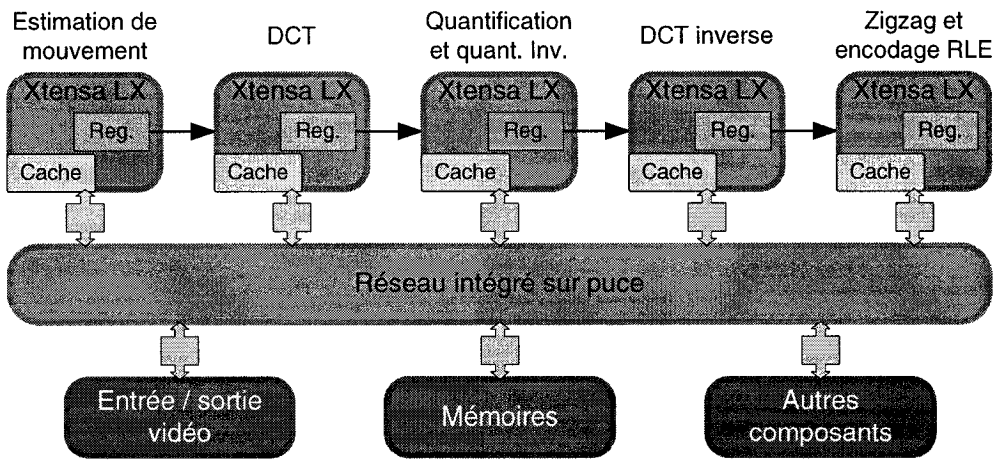

Figure 4.6 Distribution de type flot de données avec les Xtensa LX

de passer à la suivante. Pour la même plateforme, un autre type de distribution a été suggéré où l'on effectue toute les opérations d'encodage sur le même macrobloc avant d'écrire le résultat [30]. Le partage du calcul par les processeurs se fait toujours au niveau macrobloc, mais les opérations ne sont pas effectuées dans le même ordre. Cette méthode possède l'avantage de réduire le nombre de lectures à la mémoire principale puisque chaque macrobloc demeure plus longtemps dans la mémoire du processeur. Cette distribution, comme celle utilisée pour cette recherche, utilise des processeurs homogènes puisque chaque processeur doit être optimisé pour chacune des étapes de l'encodage.

Une autre approche potentielle est d'utiliser les ports, ou queues, du Xtensa LX (voir section 1.2.1.2) pour produire une distribution de type flot de données. La figure 4.6 illustre l'idée d'une telle plateforme. Un premier processeur effectue les opérations d'estimation de mouvement sur un macrobloc, et lorsqu'il a terminé il écrit ce macrobloc sur un port TIE. Le macrobloc est récupéré par le processeur suivant qui effectue l'opération de DCT, pendant que le premier effectue l'estimation de mouvement sur un deuxième macrobloc, et le passe au processeur suivant. Cette chaîne ce poursuit jusqu'à ce que toutes les opérations soient effectuées. Comme dans le cas précédent, cette approche possède l'avantage de limiter les transferts sur le canal

de communication puisque les macroblocs transigent directement d'un processeur à un autre. Un avantage supplémentaire, par contre, est que chaque processeur n'effectue qu'une seule opération de l'encodage, donc ceux-ci peuvent se permettre d'implanter seulement les instructions spécialisées qui leurs seront utiles, réduisant la taille de chaque processeur et donnant l'occasion d'y inclure des instructions plus performantes.

CONCLUSION ET TRAVAUX FUTURS

Avec la taille du transistor qui ne cesse de diminuer, la conception de systèmes sur puce devient de plus en plus complexe au fil des années. Par ailleurs, les besoins toujours croissants du marché forcent les développeurs de systèmes embarqués à se tourner vers d'autres techniques pour rencontrer leurs contraintes de performance, de coûts et de temps de développement.

Une des options à laquelle ont accès les développeurs est l'utilisation de processeurs configurables. Cette nouvelle classe de processeurs se veut un compromis entre la performance offerte par les coprocesseurs dédiés et la flexibilité offerte par les ASIP. Ceux-ci offrent des avantages à trois niveaux : extension du jeu d'instruction, sélection de blocs prédéfinis et paramétrage. La sélection de blocs prédéfinis et le paramétrage permettent d'ajuster certains paramètres architecturaux aux besoins de la plateforme (e.g. cache, bus, MAC). Par contre, l'avantage principal de ces processeurs réside dans la possibilité d'étendre le jeu d'instructions standard avec des instructions spécialisées conçues par le développeur ou déterminées automatiquement par un outil de conception. Cette propriété permet d'accélérer grandement une application, avec quelques instructions judicieusement choisies, tout en maintenant un bon niveau de flexibilité puisque nous n'utilisons que des composants logiciels.

Lors de cette recherche, nous avons montré les avantages d'utiliser un processeur configurable pour accélérer une application d'encodage MPEG-4. Les instructions réalisées permettent des gains à plusieurs niveaux. En premier lieu, l'ajout de registres spécialisés au processeur permet de couper une large part des transferts entre la cache et les registres généraux du processeur. Pour certaines fonctions qui manipulent fréquemment un très grand nombre de données, comme la DCT, cet ajout s'avère très efficace. Malheureusement, les registres spécialisés occupent une très grande surface

sur la version du Xtensa qui fut utilisée, ce qui limite leur utilisation. Un second point sur lequel il est possible de réaliser des gains appréciables est l'ajout d'extensions SIMD. Ce type d'instruction permet de réaliser x opérations mathématiques sur x groupes de données différents. Ainsi, une simple instruction SIMD peut effectuer, par exemple, quatre multiplications sur quatre paires de variables. Pour l'encodage MPEG-4, ce type d'extension procure de bons résultats pour plusieurs fonctions dont la DCT, SAD et addition de bloc. Les extensions SIMD viennent reproduire une des principales forces des processeurs DSP, sans pour autant nécessiter toutes leurs fonctionnalités supplémentaires, par exemple une unité de traitement en point flottant qui n'aurait aucune utilité pour l'application étudiée.

Un défi auquel sont confrontés les développeurs de systèmes sur puce est l'intégration de plusieurs composants ensemble. Le nombre d'éléments reliés augmente et les plateformes récentes utilisent souvent plusieurs processeurs pour distribuer le traitement. La communication et synchronisation entre tous ces composants peut s'avérer un problème de taille. Pour le résoudre, plusieurs langages de conception matérielle à haut niveau d'abstraction ont été développés. Ces langages permettent de masquer plusieurs détails d'implémentation et donc facilitent le développement et accélèrent le délai de conception. Lors de cette recherche, nous avons utilisé un de ces langages, nommé SystemC, qui est une bibliothèque C++ permettant de modéliser du matériel. Puisque cette bibliothèque possède toutes les caractéristiques du C++, il est possible d'encapsuler des modules, paquets et canaux de communication à l'intérieur de classes et de gérer leurs relations par de simples appels de fonction décrites à haut niveau d'abstraction. Pour faciliter davantage la conception de systèmes multiprocesseurs, des plateformes de développement à haut niveau d'abstraction ont également été développées. Ces plateformes fournissent généralement un éventail de composants (e.g. processeurs, mémoires, canaux de communication) pouvant être connectés ensemble pour former une plateforme complète. Par ailleurs, celles-ci offrent parfois de générer automatiquement les interfaces entre les composants, facilitant

grandement le travail d'intégration.

Lors de cette recherche, nous avons utilisé la plateforme de développement à haut niveau StepNP. Une des tâches importantes ayant été réalisées est l'intégration d'un modèle simulable du processeur Xtensa à la plateforme StepNP. Bien que StepNP soit réalisé à partir de la bibliothèque SystemC et que Tensilica fournisse une interface en langage C de son processeur, cette tâche fut beaucoup plus complexe qu'attendue. Les outils de Tensilica sont très bien conçus si l'on respecte toutes les spécifications données. Malheureusement, dès que l'on tente de modifier le fonctionnement, les outils semblent mal adaptés et la documentation est très incomplète. Néanmoins, un modèle fonctionnel du Xtensa a tout de même été intégré à la plateforme StepNP.

La dernière grande étape de cette recherche a été la simulation d'une plateforme multiprocesseurs d'encodage MPEG-4 sur StepNP. Pour nous aider dans cette tâche, StepNP fourni deux modèles de programmation multiprocesseurs à haut niveau, nommés SMP et DSOC. Ces deux modèles permettent de distribuer une application sur plusieurs éléments de calcul homogènes et hétérogènes respectivement. La présence d'une couche logicielle, faisant figure de système d'exploitation, permet de masquer la grande majorité des détails d'implémentation. Ainsi, l'utilisation des modèles SMP et DSOC se rapproche grandement d'un simple appel de fonction. De plus, l'utilisation d'accélérateurs matériels chargés d'une partie du traitement relié à SMP ou DSOC permet de lancer des processus parallèles avec seulement quelques instructions supplémentaires. À l'aide de notre implémentation de l'application d'encodage MPEG-4 sur StepNP, nous avons pu constater que l'utilisation d'instructions spécialisées procure de bons gains par rapport à une solution n'utilisant aucun accélérateur. Malheureusement, ces gains demeurent inférieurs à ceux obtenus en utilisant des coprocesseurs dédiés. Par contre, un avantage d'utiliser les instructions spécialisées est qu'elles permettent d'obtenir une meilleure accélération lorsque le nombre de processeurs augmente.

Finalement, l'utilisation de processeurs configurables et de systèmes multiprocesseurs ouvre la voie à plusieurs autres projets de recherche potentiels. Un élément qui serait très intéressant à évaluer est la génération automatique d'instructions spécialisées. Le Xtensa LX permet de la faire automatiquement et certains algorithmes plus généraux ont déjà été proposés à cet effet. De plus, sur le sujet du Xtensa LX, son utilisation dans une plateforme multiprocesseurs hétérogènes serait une suite très intéressante à ce projet de recherche. Avec ses interfaces directes entre processeurs, le Xtensa LX permettrait d'obtenir une plateforme où chaque processeur est spécialisé pour une tâche spécifique, réduisant ainsi la surface requise et augmentant l'efficacité des instructions spécialisées. De plus, avec les instructions FLIX, le Xtensa LX offre un niveau de parallélisme supplémentaire que l'on ne peut atteindre avec les versions antérieures du processeur.

RÉFÉRENCES

[1] Site web de la compagnie CoWare. Consulté le 13 mai 2006, tiré de http://www.coware.com/.

[2] Site web de la compagnie Poseidon Systems. Consulté le 6 juin 2006, tiré de http://www.poseidon-systems.com/.

[3] Site web de la compagnie Tensilica. Consulté le 8 septembre 2006, tiré de http://www.tensilica.com/.

[4] Alexander, P. et Kong, C. (2001). Rosetta: semantic support for model-centered systems-level design. *Computer*, *34*(11), 64–70.

[5] Ansari, A., Ryser, P. et Dan, I. (2005). Accelerated System Performance with APU-Enhanced Processing. *Xcell Journal*, (52), 36–39.

[6] Arnold, J. M. (2005). S5: the architecture and development flow of a software configurable processor. *Proceedings of the 2005 IEEE International Conference on Field-Programmable Technology.* (pp. 121–128).

[7] August, D., Keutzer, K., Malik, S. et Newton, R. (2000). Programmable ASICs to reduce costs. *EETimes.* Consulté le 17 Juillet 2006, tiré de http://www.eet.com/story/OEG20001120S0030.

[8] Benini, L., Bertozzi, D., Bruni, D., Drago, N., Fummi, F. et Poncino, M. (2003). SystemC cosimulation and emulation of multiprocessor SoC designs. *Computer*, *36*(4), 53–59.

[9] Benini, L. et De Micheli, G. (2002). Networks on chip: a new paradigm for systems on chip design. *Proceedings of the 2002 Design, Automation and Test in Europe Conference and Exhibition.* (pp. 418–419). Paris, France.

[10] Benny, O., Rondonneau, M., Chevalier, J., Bois, G., Aboulhamid, E. M. et Boyer, F. R. (2004). SoC software refinement approach for a SystemC platform. *Using Hardware Design and Verification Languages (DVCON2004)*. San Jose, USA.

[11] Berkeley Design Technology Inc. (2005). An Independent Analysis of the Tensilica Xtensa LX Processor with Vectra LX.

[12] Bernstein, A., Burton, M. et Ghenassia, F. (2004). How to bridge the abstraction gap in system level modeling and design. *IEEE/ACM International Conference on Computer Aided Design (ICCAD-2004)*. (pp. 910–914).

[13] Burger, D. et Goodman, J. R. (2004). Billion-transistor architectures: there and back again. *Computer*, *37*(3), 22–28. 0018-9162.

[14] Cai, L. et Gajski, D. (2003). Transaction level modeling: an overview. *First IEEE/ACM/IFIP International Conference on Hardware/Software Codesign and System Synthesis*. (pp. 19–24).

[15] Callahan, T. J., Hauser, J. R. et Wawrzynek, J. (2000). The Garp architecture and C compiler. *Computer*, *33*(4), 62–69.

[16] Canadian Microelectronics Corporation (CMC). (2004). StepNP Reference Manual Version 1.0, 165 pages.

[17] Cesário, W. O., Baghdadi, A., Gauthier, L., Lyonnard, D., Nicolescu, G., Paviot, Y., Yoo, S., Jerraya, A. A. et Diaz-Nava, M. (2002). Component-based design approach for multicore SoCs. *Proceedings of the 39th conference on Design automation*. (pp. 789–794). New Orleans, Louisiana, USA: ACM Press.

[18] Cesário, W. O., Lyonnard, D., Nicolescu, G., Paviot, Y., Sungjoo, Y., Jerraya, A. A., Gauthier, L. et Diaz-Nava, M. (2002). Multiprocessor SoC platforms: a component-based design approach. *Design & Test of Computers, IEEE*, *19*(6), 52–63.

[19] Chevalier, J., de Nanclas, M., Filion, L., Benny, O., Rondonneau, M., Bois, G. et El Mostapha, A. (2006). A SystemC refinement methodology for embedded software. *Design & Test of Computers, IEEE*, *23*(2), 148–158.

[20] Clark, N. T., Zhong, H. et Mahlke, S. A. (2005). Automated custom instruction generation for domain-specific processor acceleration. *IEEE Transactions on Computers*, *54*(10), 1258–1270.

[21] Cong, J., Fan, Y., Han, G. et Zhang, Z. (2004). Application-specific instruction generation for configurable processor architectures. *Proceedings of the 2004 ACM/SIGDA 12th international symposium on Field programmable gate arrays*. (pp. 183–189). Monterey, California, USA: ACM Press.

[22] Cravotta, R. (2004). Accelerate your performance. *EDN*, 50–62. Consulté le 8 septembre 2006, tiré de http://www.edn.com/article/CA476908.html.

[23] Deslaurier, F. (2005). Modélisation d'un reseau intégré sur puce extensible basé sur une architecture en anneau. Mémoire de maîtrise, École Polytechnique de Montréal, Québec, Canada.

[24] Drucker, L. (2003). SystemC Verification Library speeds transaction-based verification. *EETimes*. Consulté le 13 novembre 2006, tiré de http://www.eetimes.com/story/OEG20030214S0042.

[25] Dutt, N. et Kiyoung, C. (2003). Configurable processors for embedded computing. *Computer*, *36*(1), 120–123.

[26] Dziri, M. A., Samet, F., Wagner, F. R., Cesário, W. O. et Jerraya, A. A. (2003). Combining architecture exploration and a path to implementation to build a complete SoC design flow from system specification to RTL. *Proceedings of the 2003 Design Automation Conference*. (pp. 219–224).

[27] Fitzpatrick, T. (2003). Assertions in SystemVerilog: A Unified Language for More Efficient Verification.

[28] Fitzpatrick, T. (2004). SystemVerilog for VHDL users. *Design, Automation and Test in Europe Conference and Exhibition.* (Vol. 2, pp. 1334–1339).

[29] Fujita, M. et Nakamura, H. (2001). The standard SpecC language. *14th International Symposium on System Synthesis.* (pp. 81–86).

[30] Gagne, V. (2006). Memoire de Vincent. Mémoire de maîtrise, Université de Montréal, Québec, Canada.

[31] Goering, R. (2006). Dataquest to EDA: 'It's the software, stupid'. *EETimes.* Consulté le 22 mai 2006, tiré de http://www.eetimes.com/conf/dac/showArticle.jhtml?articleId=191000150&kc=2443.

[32] Gonzalez, R. E. (2000). Xtensa: a configurable and extensible processor. *Micro, IEEE, 20*(2), 60–70.

[33] Hellestrand, G. (2005). The engineering of supersystems. *Computer, 38*(1), 103–105.

[34] Henkel, J. (2003). Closing the SoC design gap. *Computer, 36*(9), 119–121.

[35] Iranpour, A. R. et Kuchcinski, K. (2004). Evaluation of SIMD architecture enhancement in embedded processors for MPEG-4. *Euromicro Symposium on Digital System Design.* (pp. 262–269).

[36] Jerraya, A., Tenhunen, H. et Wolf, W. (2005). Guest Editors' Introduction: Multiprocessor Systems-on-Chips. *Computer, 38*(7), 36–40.

[37] Lapalme, J., Aboulhamid, E. M., Nicolescu, G., Charest, L., Boyer, F. R., David, J. P. et Bois, G. (2004). .NET Framework – A Solution for the Next Generation

Tools for System-Level Modeling and Simulation. *Proceedings of the conference on Design, automation and test in Europe.* (Vol. 1, pp. 732–733). IEEE Computer Society.

[38] Lavigueur, B. (2005). Multitraitement et processeurs configurables sur une plate-forme de haut niveau. Mémoire de maîtrise, École Polytechnique de Montréal, Québec, Canada.

[39] Leibson, S. et Kim, J. (2005). Configurable processors: a new era in chip design. *Computer, 38*(7), 51–59.

[40] Maniwa, T. (2004). Focus Report: Electronic System-Level (ESL) Tools. A bolt-on to RTL or a new methodology? *Chip Design Magazine.* Consulté le 29 avril 2006, tiré de http://www.chipdesignmag.com/display.php?articleId=23&issueId=4.

[41] Martin, G. ESL Requirements for Configurable Processor-based Embedded System Design. *Design and Reuse.* Consulté le 27 novembre 2005, tiré de http://www.us.design-reuse.com/articles/article12444.html.

[42] Mbaye, M., Belanger, N., Savaria, Y. et Pierre, S. (2005). Application specific instruction-set processor generation for video processing based on loop optimization. *IEEE International Symposium on Circuits and Systems (ISCAS).* (Vol. 4, pp. 3515–3518).

[43] Mihal, A., Kulkarni, C., Moskewicz, M., Tsai, M., Shah, N., Weber, S., Yujia, J., Keutzer, K., Vissers, K., Sauer, C. et Malik, S. (2002). Developing architectural platforms: a disciplined approach. *Design & Test of Computers, IEEE, 19*(6), 6–16.

[44] Moretti, G. (2005). Design complexity requires system-level design. *EDN.* Consulté le 8 mars 2006, tiré de http://www.edn.com/article/CA505068.html.

[45] Nava, M. D., Blouet, P., Teninge, P., Coppola, M., Ben-Ismail, T., Picchiottino, S. et Wilson, R. (2005). An open platform for developing multiprocessor SoCs. *Computer*, *38*(7), 60–67.

[46] OSCI. (2005). SystemC Language Reference Manual.

[47] Panda, P. R. (2001). SystemC - a modeling platform supporting multiple design abstractions. *14th International Symposium on System Synthesis.* (pp. 75–80).

[48] Paulin, P. G., Pilkington, C. et Bensoudane, E. (2002). StepNP: a system-level exploration platform for network processors. *Design & Test of Computers, IEEE*, *19*(6), 17–26.

[49] Paulin, P. G., Pilkington, C., Langevin, M., Bensoudane, E., Benny, O., Lyonnard, D., Lavigueur, B. et Lo, D. (2006). Distributed object models for multi-processor SoC's, with application to low-power multimedia wireless systems. *Proceedings of the conference on Design, automation and test in Europe: Proceedings.* (pp. 482–487).

[50] Paulin, P. G., Pilkington, C., Langevin, M., Bensoudane, E., Lyonnard, D., Benny, O., Lavigueur, B., Lo, D., Beltrame, G., Gagne, V. et Nicolescu, G. (2006). Parallel Programming Models for a Multi-Processor SoC Platform Applied to Networking and Multimedia. *IEEE Transactions on VLSI*, 48–53.

[51] Paulin, P. G., Pilkington, C., Langevin, M., Bensoudane, E. et Nicolescu, G. (2004). Parallel programming models for a multi-processor SoC platform applied to high-speed traffic management. *Hardware/Software Codesign and System Synthesis.* (pp. 48–53).

[52] Pees, S., Hoffmann, A., Zivojnovic, V. et Meyr, H. (1999). LISA-machine description language for cycle-accurate models of programmable DSP architectures. *Design Automation Conference.* (pp. 933–938).

[53] Quinn, D., Lavigueur, B., Bois, G. et Aboulhamid, M. (2004). A system level exploration platform and methodology for network applications based on configurable processors. *Proceedings of Design, Automation and Test in Europe.* (Vol. 1, pp. 364–369).

[54] Richardson, I. vcodex. Consulté le 12 Juillet 2006, tiré de http://www.vcodex.com/.

[55] Samson, P. (2006). Abstraction de la synchronisation et des communications dans une stratégie de co-design logiciel/matériel en vue du raffinement sur une plateforme (SoC) multiprocesseur hétérogène. Mémoire de maîtrise, École Polytechnique de Montréal, Québec, Canada. Titre prévu.

[56] Schirrmeister, F. et Sangiovanni-Vincentelli, A. (2001). Virtual component co-design-applying function architecture co-design to automotive applications. *Vehicle Electronics Conference.* (pp. 221–226).

[57] Tensilica. Increasing Computational Performance Through FLIX (Flexible Length Instruction Extensions). Consulté le 25 mai 2006, tiré de http://www.tensilica.com/products/WP_flix.htm.

[58] Tensilica. (2003). Tensilica Instruction Extension (TIE) Language User's Guide.

[59] Tohara, T. (2005). A New Kind of Processor Interface for a System-on-Chip Processor with TIE Ports and TIE Queues of Xtensa LX. *Innovative Architecture for Future Generation High-Performance Processors and Systems.* (pp. 72–79).

[60] Turley, J. (2005). Survey says: software tools more important than chips. *Embedded.com.* Consulté le 6 mai 2006, tiré de http://www.embedded.com/showArticle.jhtml?articleID=160700620.

[61] van der Wolf, P., de Kock, E., Henriksson, T., Kruijtzer, W. et Essink, G. (2004). Design and programming of embedded multiprocessors: an interface-

centric approach. *International Conference on Hardware/Software Codesign and System Synthesis.* (pp. 206–217).

ANNEXE I

PLATEFORME MATÉRIELLE ET OUTILS SOC DE STEPNP

I.1 Environnement de développement matériel

L'environnement de développement matériel permet de construire une plateforme complète reliant ensemble des composants décrits en SystemC. StepNP fournit une large banque de composants prédéfinis pouvant être utilisés, mais il est également possible d'en créer des nouveaux ou encore de simplement étendre les fonctionnalités de ceux déjà présent. De plus, les composants de StepNP possèdent une interface de communication simple permettant de rapidement et facilement brancher ceux-ci ensemble.

Processeurs. La version de base de StepNP fournit deux modèles de processeurs soit celui du ARM et du PowerPC. Afin de rendre la présence d'un ISS transparente à l'usager, tous les modèles de processeurs sont encapsulés dans une enveloppe SystemC. Les modèles de ces deux processeurs possèdent des capacités de multi-traitement matériel permettant de masquer la latence des communications. Cette propriété est réalisée en utilisant plusieurs instances de l'ISS par processeur qui s'exécutent chacune à tour de rôle. Présentement, l'ordonnancement des processus suit une politique de séquencement à la ronde (*round robin*) mais l'équipe de StepNP vise à permettre des ordonnancements plus évolués (e.g. RMA, DMA, EDF). La présence de fonctions de rappel inclusent avec les ISS permet de rediriger les accès mémoire vers le reste de la plateforme SystemC. Finalement, comme discuté à la section 3.2, un modèle du processeur Xtensa a également été intégré à la plateforme StepNP pour inclure les processeurs configurables au choix d'exploration.

Interconnections. Pour connecter ensemble tous les composants de la bibliothèque, il va de soit qu'un ensemble de canaux de communication est nécessaire. StepNP en fournit un éventail prêt à être utilisé. Le niveau d'abstraction de ces canaux varie entre purement fonctionnel, fonctionnel avec délais ou encore précis au cycle près et tous supportent les transactions différées (*split*). La bibliothèque de canaux inclut entre autres :

- Un canal à branchements points à points (*crossbar*), celui-ci pouvant être configuré avec ou sans délai.
- Un modèle du Hot Potato (en français patate chaude). Ce canal de communication est basé sur une maille à deux dimensions.
- Le RoC (*rotator on chip*, rotateur sur puce) classique. Ce nouveau modèle de NoC (*network on chip*, réseau sur puce) a été développé ici même à l'École Polytechnique de Montréal. Il implémente une architecture similaire à celle en anneau (*token ring*) utilisée dans les réseaux d'ordinateurs [23].
- Le StBus. Bien qu'il ne fasse pas partie de la distribution de base de StepNP, un modèle précis au cycle près de ce bus, commercialisé par STMicroelectronics, a été intégré à la plateforme. Ce bus se veut un compromis entre la rapidité d'un *crossbar* et le faible coût d'un bus simple en permettant à l'utilisateur de configurer le nombre de ressources qui peuvent partager une connexion du bus.

StepNP dispose également d'un canal d'instrumentation, nommé PerNoc (*performance NoC*), qui permet de recueillir certaines informations utiles sur les communications entre chaque maître et esclave, comme par exemple le nombre de transactions, le délai des transactions et la bande passante.

Les modules matériels de StepNP communiquent ensemble en utilisant le protocole SOCP (*SystemC OCP (Open Core Protocol)*). Comme son nom l'indique, ce

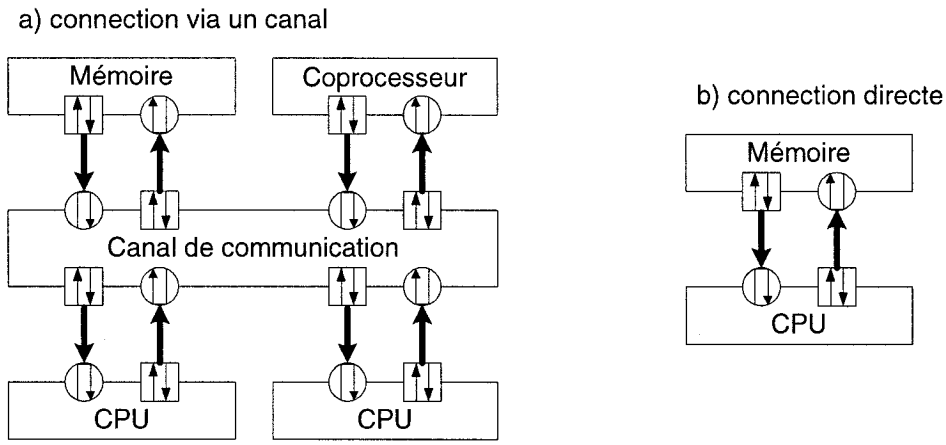

Figure I.1 Communications dans StepNP

protocole est une adaptation du protocole OCP et permet de connecter ensemble des composants virtuels décrits en SystemC. Puisque les communications sont modélisées au niveau TLM, les différents signaux du protocole OCP sont encapsulés dans une structure C pour fin de traitement. Les communications de StepNP suivent un mécanisme maître-esclave, les maîtres connectent leur port à une interface esclave et les esclave leur port à une interface maître. Le canal n'est qu'un module SystemC possédant plusieurs ports maîtres et esclave, tel que montré à la figure I.1a. Il est également possible de connecter directement un maître et un esclave sans passer par un canal comme illustré à la figure I.1b.

Ponts. StepNP fourni aux processeurs un mécanisme de ponts pouvant s'insérer entre eux et le canal de communication (ou autre esclave SOCP). Lorsqu'un processeur effectue un accès mémoire, il parcourt la liste de ses ponts attachés et tente d'en trouver un qui peut répondre à la requête. Si un des ponts répond à la requête, l'accès mémoire est terminé, sinon il est transféré sur le canal SOCP et sera traité par un composant de la plateforme. Puisque les ponts sont également des maîtres SOCP, ils peuvent également passer la requête à la plateforme (avec possiblement des modifications) avant de répondre au processeur. La liste suivante donne un aperçu des fonctions effectuées par les ponts :

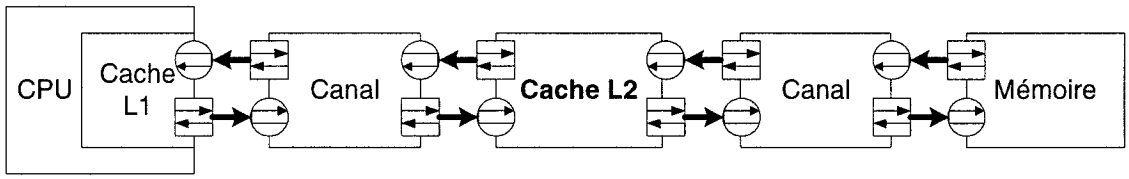

Figure I.2 Cache L2 dans StepNP

- Accès non alignés. Effectue deux accès mémoire et concatène le résultat
- Mémoire virtuelle. Petite mémoire attachée directement au processeur, évitant l'utilisation d'un module supplémentaire pour la mémoire.
- Zone magique. Ce pont effectue des traitements propres à une simulation, comme la suspension et l'arrêt de la simulation ou l'affichage de message à l'écran, il ne correspond à rien de réel. Pour ce faire, ce pont utilise une zone mémoire spéciale qui lui est dédiée.

Mémoires et Caches. StepNP possède, en plus de la mémoire virtuelle décrite à la section sur les ponts, un modèle simple de mémoire implémentée sous forme de tableau encapsulé dans un esclave SOCP.

Les caches quant à elles sont pour la plupart implémentées en utilisant les ponts de StepNP. Lors d'un accès mémoire, la cache vérifie si la donnée est présente, si oui elle la retourne, si non elle va la chercher sur le canal et la retourne ensuite. Il existe également une cache L2, non disponible dans la distribution de base de StepNP, qui agit également comme pont entre deux canaux, comme illustré à la figure I.2. Cette cache agit à la fois à titre d'esclave sur le canal côté processeur et comme maître sur le canal côté mémoire. Cette implémentation peut être utilisée dans une plateforme hiérarchique où le système complet est constitué de plusieurs sous systèmes et où tous les sous système se partagent une mémoire centrale commune.

I.2 Suite d'outils SoC

La simulation d'un système sur puce ne possède pas un grand intérêt sans la possibilité d'analyser, contrôler et déverminer la plateforme. Pour répondre à ce besoin, StepNP dispose d'une suite d'outils externes capable de se connecter à la plateforme matérielle et effectuer ces fonctions.

SIDL. Le protocole SIDL (*SystemC Interface Definition Language*, Langage de définition d'interfaces pour SystemC) a été intégré à StepNP dans le but de permettre aux éléments de la plateforme matérielle de communiquer avec les outils externes. Ceci nous permet, dans un premier temps, de visualiser les données de la plateforme et favorise le déverminage. Dans un deuxième temps, ceci donne la possibilité aux outils externe de contrôler la simulation. Ce protocole est inspiré d'autres IDL bien connus tels CORBA et DCOM, par contre il est plus léger et ne s'encombre pas de fonctionnalités comme l'authentification et adopte une syntaxe plus près de celle de SystemC. Les interfaces en SIDL sont spécifiées sous forme d'une classe virtuelle pure en C++ et de structures sans aucune fonction membre. Les classes abstraites ainsi décrites sont compilées par le compilateur SIDL qui génère les fichiers source pour le client et le serveur dans le langage choisi. Les langages supportés actuellement sont C, C++, Java et Python et celui-ci n'a pas à être le même pour le client et le serveur. De plus, l'équipe de développement de StepNP projette de supporter Tcl et Perl dans leurs développements futurs. Une fois les fichiers source générés, il suffit de les compiler et les exécutables pourront communiquer ensemble via des RPC (*Remote Procedure Call*, appel de procédure à distance).

Sondes. Les sondes sont un outil permettant l'introspection de variables présentes dans les éléments de la plateforme matérielle. De manière plus concrète, les sondes sont des classes génériques (*template C++*) et si une sonde nommé X possède comme paramètre le type T (la déclaration serait donc `probe_value<T> X;`), cela signifie

que X peut être utilisé comme s'il s'agissait d'une variable de type T. Par contre, la variable X peut maintenant communiquer avec les outils externe ce qui permet de [16] :

- Lire et écrire sur cette variable à partir des outils externes
- Obtenir l'historique des valeurs de la variable
- Placer un point d'interruption en fonction de la valeur de cette variable
- Tracer cette variable avec la fonction `sc_trace` de SystemC

Interface graphique. Pour favoriser le contrôle et la visualisation de la plateforme, StepNP fournit une interface graphique, nommée SocMon (*SoC Monitor*, moniteur de systèmes sur puce), réalisée en Java. Un aperçu visuel de SocMon est présenté à la figure 3.1c. SocMon fourni plusieurs vues disponibles à l'utilisateur présentant chacune de l'information complémentaire permettant d'effectuer une analyse détaillée de la plateforme sous test. Malheureusement, quelques fonctionnalités décrites ne fonctionnent que de manière partielle et du travail demeure nécessaire pour que SocMon soit à la hauteur des aspiration de ses développeurs. Les différentes vues de SocMon sont décrites ci-dessous.

La vue de ligne temporelle utilise les sondes décrites précédemment pour présenter une vue graphique de leur évolution dans le temps.

La vue de modèle de programmeur permet d'obtenir des informations sur les processeurs de la plateforme, notamment sur leurs registres.

La vue de contrôle et composants inspecte l'ensemble de la plateforme et extrait la hiérarchie des composants qui y sont présents.

Le vue du code source montre le code s'exécutant sur chaque processus de chaque processeurs. Il est également possible de spécifier le temps auquel on désire voir le code s'exécutant sur un processeur et pas nécessairement l'exécution courrante. Un point très intéressant est que StepNP inclut une

version temporelle de GDB, nommée TGDB pour *temporal GDB*, qui offre les mêmes fonctionnalités que GDB mais avec la possibilité de reculer dans le temps.

La vue assembleur offre essentiellement les mêmes caractéristiques que la vue du code source mais, comme son nom l'indique, on y voit le code assembleur plutôt que le code haut niveau.

La vue de scripts permet d'éditer et lancer des scripts de configuration. Les scripts de configuration seront élaborés plus en détail lors de la génération dynamique.

Les vues personnalisées sont réalisées par l'utilisateur de StepNP et permettent de mieux adapter les outils aux besoins de la plateforme. Ces vues prennent la forme de plugiciels *plugins* Java que SocMon charge dynamiquement en cours d'exécution.

Génération dynamique. En plus de fournir de puissants outils de visualisation, StepNP procure un mécanisme facilitant grandement la génération et la configuration d'une plateforme. Normalement, pour générer une plateforme en SystemC il est nécessaire de créer un programme exécutable en C++ dans lequel on instancie, configure et connecte tous les composants de la plateforme que l'on désire simuler, et chaque plateforme possède sont propre exécutable. SocGen (*SoC Generation*, génération d'un système sur puce) est un outil de StepNP qui permet d'éliminer cette restriction et place la génération de la plateforme à l'étape d'exécution plutôt qu'à celle de compilation. À l'aide de SocGen, toutes les plateformes utilisent un exécutable commun implémentant un serveur StepNP. La configuration se fait par un script externe qui exécute les opérations suivantes :

1. Connection au serveur
2. Chargement dynamique de composants
3. Création et configuration des composants voulus
4. Connexion des composants ensemble

5. Départ de la simulation

SocGen réutilise le protocole SIDL défini précédemment où le serveur StepNP est un serveur SIDL C++ et le script est un client SIDL réalisé dans la langage de choix du développeur. Puisque la communication entre les deux est générée automatiquement par le compilateur SIDL et que les fonctionnalités du serveur sont déjà implantées, il ne reste à l'utilisateur qu'à appeler les fonctions appropriées de l'interface pour configurer la plateforme.

Comme on peut s'en douter, les avantages d'un outil de configuration haut niveau tel SocGen sont nombreux [16] :

- Procure une plateforme extensible avec seulement un fichier exécutable.
- La création de nouveaux composants compatibles avec SocGen est facile.
- Il n'est pas nécessaire d'avoir une connaissance approfondie de SystemC pour créer et simuler une plateforme puisque ces opérations sont effectuées en Python, Java ou C++.
- La configuration et le contrôle sont simplifiés puisqu'ils sont fait dans un script plutôt qu'un programme.